云计算

通俗讲义 | 第3版 |

王良明◎著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

VISIT…

内 容 简 介

本书力求简明扼要地阐述云计算的基本概念，让非IT专业人士也能轻松看懂这一现在人人都能接触（以后程度会更深）的事物——云计算。本书遵循一条由感性到理性、由浅入深的主线展开：首先采用情景描述，让外行的人对云计算也能产生感性认识；其次从计算机的基本知识开始逐步引出云计算的概念，使人产生理性认识；接着列举一系列生活中的典型云应用例子，让人们明白云计算给生活带来的确切好处；最后采用OpenStack搭建单节点和三节点方案让人们感受组建云系统的过程，以达到抛砖引玉的目的。

本书可作为高等院校学生教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

云计算通俗讲义 / 王良明著. —3 版. —北京：电子工业出版社，2019.3
ISBN 978-7-121-35942-2

Ⅰ. ①云… Ⅱ. ①王… Ⅲ. ①云计算－高等学校－教材 Ⅳ. ①TP393.027

中国版本图书馆 CIP 数据核字（2019）第 011661 号

策划编辑：孙学瑛
责任编辑：孙学瑛
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：720×1000　1/16　印张：22.75　字数：316 千字
版　　次：2015 年 11 月第 1 版
　　2019 年 3 月第 3 版
印　　次：2019 年 3 月第 1 次印刷
定　　价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前 言

从 2011 年开始，云计算成为国内线上线下的热谈，政府主导的 5 个云计算中心浮现在国人的眼前，各种新型云计算公司如雨后春笋般涌现，许多非互联网公司也纷纷增加与云计算有关的产品线，各种云存储产品容量节节攀升。伴随而来的是云计算书籍的大量涌现，在书店和网店都占据着显著的位置。笔者从 2006 年开始涉足云计算，自然不会错过这些新涌现的书籍，但翻了很多这类书之后发现，这些书中堆满了各种专业术语和英文单词，很多内容都是直译外国文档资料，但都没有说清楚什么是云计算。在云计算应用风起云涌的当下，云计算、物联网和大数据的概念异常混乱，而且诸如智慧小区、智慧城市和智慧地球等新概念又不断被炒出来，更是让人“不识庐山真面目”了。

在中国，云计算呈现冰火两重天的怪象：这边厢，供应商们个个摩拳擦掌、热情高涨，大家恨不得从“万亿云计算市场”蛋糕中分得一大块，却鲜有人脚踏实地做产品；那边厢，用户们迷茫、观望者众多，大家都牢牢捂紧各自的钱袋，怯怯地问：“云计算到底是什么？能给我带来什么好处？”

笔者深刻地感受到，必须要向民众简单明了地讲清楚云计算的概念，同时对各种热炒的新概念做一个全面的梳理，正本清源，才能使得国内云计算

产业步入良性发展的轨道。

本书遵循由感性认识到理性认识、由浅入深的原则展开讨论。第 1 章“云”畅想，以未来一家人的衣食住行为情节来描述美好的云生活，使得读者对云计算有一个初步的感性认识。第 2 章“云”概念，从最基本的计算机知识开始，逐步引入云计算的几个主要特征，最后阐述云计算的概念就水到渠成了。第 3 章“云”架构，重点介绍美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）发布的云计算架构模型。第 4 章“云”组件，从整体上梳理云计算的组成结构，并阐述各个组成部分对应的开源和商业解决方案。第 5 章“云”技术，介绍组建云计算中心需要用到的技术，向读者表明云计算只不过就是传统 IT 技术的集大成者，并没有什么神秘的新技术。第 6 章“云”安全，从多个角度介绍云计算的安全性，意在消除部分民众认为云计算不安全的恐惧心理。第 7 章“云”应用，罗列一些生活当中最典型的云应用，目前这些云应用无一被实现，是未来的重点发展方向。第 8 章“云”实验，采用 OpenStack 分别搭建单节点和三节点 IaaS 云实验平台，步骤详细。

本书自第 1 版发行以来，收到了很多读者的反馈，于是笔者结合自己的最新教学总结，加上云计算快速发展带来的新东西，在第 2 版的基础上继续推出第 3 版。相对于第 2 版，第 3 版主要做了如下改进：第 6 章增加了当前主流云服务提供商的各自产品介绍和综合比较；第 8 章全部重写，方便读者在一台电脑上就能搭建云实验平台。

非技术人员通过阅读本书，能感性认识云计算；

企业管理者通过阅读本书，能发现商机；

IT 技术人员通过阅读本书，能建立 IT 环境的全局观，并对云计算涉及的方方面面技术产生理性认识；

云计算工程师通过阅读本书，会重新梳理大脑中原先杂乱无章的概念，且有可能将对云计算的认识提高到哲学层面；

大学生通过阅读本书，可以很好地建构自己的云计算知识体系……

总之，每个读者都能从中获益，至少对云计算不再“云里雾里”了。

我要感谢一路陪同我的妻子、女儿、父母及兄弟姐妹，他们是上帝赐予我最宝贵的财富，我们一路同行，一路欢笑。

恳求各位读者提出宝贵的意见和建议，荣誉归于你们，不胜感激。

本书一些配套的资料请浏览网站 www.veryopen.org，有意见和建议请发送邮件至 wlm@veryopen.org。

作者 于岭南·五桂山（广东理工职业学院）

2018 年 12 月 19 日

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **提交勘误：**您对书中内容的修改意见可在【提交勘误】处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与作者交流：**在页面下方【读者评论】处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/35942

目 录

第1章 “云”畅想 1

第2章 “云”概念 18

2.1 软件的概念 19

2.2 计算机系统 25

2.3 计算机网络 27

2.4 IT 系统组成 32

2.5 云计算概念 36

2.6 3 种服务模式 42

2.6.1 IaaS 43

2.6.2 PaaS 47

2.6.3 SaaS 52

2.7 4 种部署模型 59

2.7.1 私有云 60

2.7.2 社区云 62

2.7.3 公有云 64

2.7.4 混合云 65

2.8 云计算的优劣分析 68

2.8.1 情景案例 68
2.8.2 云计算的优势 73
2.8.3 云计算的劣势 74
第 3 章 “云”架构 76
3.1 云架构参考模型 76
3.2 云服务消费者 80
3.3 云服务提供商 81
3.3.1 服务编排 82
3.3.2 云服务管理 83
3.3.3 安全保障 86
3.3.4 隐私保护 88
3.4 云计算审计员 89
3.5 云服务代理商 89
3.6 云服务承运商 91
第 4 章 “云”组件 92
4.1 开源组件 95
4.1.1 开源计算项目 95
4.1.2 虚拟化 97
4.1.3 操作系统 103
4.1.4 数据库 104
4.1.5 中间件 114
4.1.6 基础服务组件 118
4.1.7 云管理工具 124
4.1.8 应用软件 139
4.2 商业组件 140
4.2.1 VMware 140
4.2.2 Citrix 141

4.2.3 Microsoft 143
第 5 章 “云”技术 145
5.1 云端技术 145
5.1.1 云布局 147
5.1.2 存储 150
5.1.3 虚拟化与容器技术 157
5.1.4 远程桌面 168
5.1.5 负载均衡 172
5.1.6 集群 174
5.1.7 容错计算 177
5.1.8 家目录漫游 181
5.1.9 租户隔离 183
5.1.10 统一身份认证 187
5.1.11 SOA 189
5.1.12 微服务 195
5.2 终端技术 195
5.2.1 终端分类 195
5.2.2 质量控制 198
5.2.3 终端项目 198
5.3 通信协议 200
第 6 章 “云”安全 212
6.1 信息管理与数据安全 213
6.1.1 云信息架构 213
6.1.2 数据打散存储 216
6.1.3 数据安全生命周期 216
6.1.4 数据安全 217
6.2 计算可用性 220

6.3 互操作性与可移植性.....223
6.3.1 互操作性.....223
6.3.2 可移植性.....224
6.3.3 实施建议.....225
6.3.4 用户自由度.....227
6.4 加密与密钥管理.....228
6.4.1 加密介绍.....229
6.4.2 云端数据库加密.....229
6.4.3 密钥管理.....230
6.5 云服务提供商.....232
第 7 章 “云”应用.....239
7.1 企业私有办公云.....240
7.2 园区云.....241
7.3 医疗云.....243
7.4 公民档案云.....244
7.5 卫生保健云.....245
7.6 教育云.....245
7.7 交通云.....247
7.8 出行云.....248
7.9 购物云.....248
7.10 农村农业云.....249
7.11 高性能计算云.....250
7.12 人工智能云.....250
第 8 章 “云”实验.....252
8.1 单节点安装.....255
8.1.1 系统设计.....255
8.1.2 安装过程.....255

8.1.3 启动一台虚拟机 279
8.1.4 安装 Horizon 282
8.2 三节点安装 284
8.2.1 系统设计 284
8.2.2 安装过程 288
8.3 OpenStack 管理 345

第 1 章

“云”畅想

早上 6:30，家庭壁挂云终端响铃并提醒：“现在是早上 6:30。室内温度 20℃，室外温度 10℃，PM2.5 指数 20，湿度 40%，有雾，适合晨练。”我家的壁挂云终端接入了人工智能公有云端，它为我们提供建议，协助我们作决定，成为我们生活中不可或缺的朋友。只不过这个全球人工智能公有云端诞生还不到两年，它的知识非常缺乏，各个国家都在不断完善其算法，并充实其知识库。

我看了一眼妻子手腕上的胎音监护仪，亮着微弱的绿光，于是起床把正在另一个房间睡觉的儿子叫醒：“该起床了，男子汉。”儿子一骨碌爬起来。昨天我们夸他是个男子汉，男子汉就该果断、说到做到、从不拖拉。我们父子二人照例到了客厅，在云终端上很快做完了体检并拍下照片，显示体重、体温，且心率、血压、心电图、脑电波正常，儿子又长高了半毫米。到今日，儿子已经拍下了近 2000 张同角度的照片，而我的同角度的照片也已经拍了快 8 年了。每次按下连续播放按钮，看到快速长大的儿子，我都会哈哈大笑；但

是看到自己慢慢老去的脸，心情就不一样了。儿子去了卫生间，我马上看到屏幕上出现了儿子的其他体能指标：体液微酸、尿液正常、大便偏干。我皱了皱眉头："这小家伙肥肉又吃多了。"然后我又查看了一下昨夜老婆的监护记录，一共有 7 条数据，看起来一切正常。儿子去穿袜子和运动鞋了，我进了卫生间，本来是不想上厕所的，只是每天起床后都要收集体检数据，这是国家监护中心要求的。我发现自己的体液也偏微酸，以前吃碱性食品多，体液一直微碱，但最近几个月经常在餐馆大吃大喝，生日 Party、节日聚会特别多。本月 12 日我被升为公司的二级荣誉员工，经常被同事拉去请客吃饭。临出门，我根据壁挂云终端上给出的稀饭配方，淘好米并加入少许螺旋藻和高粱，这样我们回来时稀饭就煮好了。

我们跑出小区大门，然后右拐，继续朝前跑过两个街区，便到了小河边。河的两岸长着参天大树，左岸两米开外铺了一条三米宽的水泥路，路的左侧同样是参天大树，在路上行走就像进了隧道。这条路是晨跑的最佳场所，我在这条路上已经跑了足足六个年头。我向前加速跑了 200 米左右后，又往回跑，遇到儿子后鼓励他一番，然后又加速向前跑……

大概 7:40，我手腕上的终端铃响了，看到"张丽华"的名字在闪动，她是老婆妊娠期的监护医生。我按下通话按钮，张阿姨如连珠炮似的声音立马响起："小王，不错啊，又在锻炼呢！哦，还带着儿子呢，我家那位就知道睡觉……我刚刚收到国家监护中心的消息，说你老婆今天下午 15:00 要生。你们做好准备，医院上午 10:00 左右安排车和医护人员过去接你老婆，床位我已经安排好了。"张阿姨就是这么爽快，说话从不拖泥带水。

国家监护中心的预产期预测软件是我所在的申云公司开发的，是我设计的预测算法。这是大数据算法的一种，这个算法具备自我完善功能，理论上只要基础数据足够，预测准确率可达 100%，目前准确率达到了 90%。国家监护中心是政府资助的非营利性机构，属于全国性的公有云计算中心，那里运行着成百上千个与国民健康有关的云软件，监护着国家十多亿公民的身体状况，肩负着各种疾病的预测预防任务。

“张阿姨，我们要准备什么东西呢？你看，我和我老婆都没有经验，自从我儿子出生之后，国家政策发生了很大的变化。”

“主要是心理准备，婴儿和产妇的用品，医院已经全部免费提供了。”

5 年前，国家施行妇女免费分娩政策，而且医疗、教育都全民免费了。由于国家健康云的成功实施，民众的日常体能监护、卫生保健都是免费的，而且体能监护还是实时的。国家监护中心会根据每个人的身体状况发送饮食建议和注意事项，并能预测人什么时候得病、会得什么病，同时给出相应的预防措施。

我展开手腕上的终端屏幕，查看老婆的监护信息，没有什么异常。她已经起床了，看来她也知道今天要临产了，显得有点紧张。我和儿子拍了一张鬼脸合照发给老婆，同时留言：“我们现在就往回跑。”

到了小区楼下的社区服务部取了早餐和老婆的营养餐，这个服务部是老婆开的，请了两个员工。服务部一方面负责小区的社会性工作（如接收妇幼保健院免费提供的孕妇营养餐，并分发给小区内的孕妇，小区内居民云终端的发放、回收和维修及幼儿鲜奶的发放等）；另一方面开展一些盈利性业务（如 3D 打印服务、快递收发等）。我们的早餐是国家监护中心根据早上的体检数据给出建议配单并通过 3D 食品打印机打印出来的，3D 食品打印就是用营养胶把食品粉末打印成各种食品的形状而已。中国是目前唯一一个能把农产品研制成粉末并做长时间保存和运输而百分之百保留营养成分的国家，目前政府把此项技术无偿提交给联合国，相信很快就会在全世界范围内推广。

吃完早饭，儿子被幼儿园校车接去上学了。上午 9:00，我参加了公司的虚拟会议，讨论了广西农业云项目的实施进展。根据国家的规划目标，再过五年，农业云要覆盖全国农村，功能包括水、土壤、空气监控，农作物种植专家指导，农产品收割、粉末研制加工、储藏及运输等，我所在的申云公司中标了广西壮族自治区农业云项目。

10:30，张阿姨等人把我老婆接到了医院，虽然我可以不去，但我还是去

了医院，其实我也帮不上什么忙，连待产室都不让进。于是，我来到病人家属休息区，把手腕上的云终端连接到桌子上的固定云终端上，然后接入公司的私有办公云，登录云中的私人计算机桌面；同时在屏幕的右上角开启一个小窗口，实时观察我老婆的体能状况。

今年年初，公司开始进入区域医疗云市场，当然不是终端医疗云应用软件，而是医疗云平台软件、中间件和远程医疗设备。我的工作是解决远程手术当中的通信协议问题。其中，网络延时问题困扰了我很长时间。远程手术对于医生来说就像腹腔镜，看着一个屏幕操作两个机器手臂，最大的区别就是腹腔镜的病人就躺在旁边的手术台上，而远程手术的病人却在另外一所医院、另外一个城市，甚至是另外一个国家。如果网络延时过大，那么医生每次移动一下机器手臂，几秒钟后才能在屏幕上看到手术刀移动，这是不可接受的，而且很危险。通过查阅网络上论文库中的文章，我逐渐总结了这样一个结论：网络延时是网络的固有属性，与网络路径上的转发设备的数量和转发速度相关，只在网络路径的两端是无法改善网络延时的。

中午，3D 打印服务部给我送来了营养餐，这是公司的一项福利，员工周一至周五的午餐免费提供，菜谱由国家监护中心定制，分量也由它预测，保证每个人每顿饭都能吃到可口且适量的饭菜。我顺便查看了一下儿子上午的活动轨迹，目前保持不动，应该在睡午觉。活动轨迹显示，这小家伙整个上午几乎没停过。咦，怎么有两根线离开学校了？线的另一端是博物馆，应该是去参观博物馆了。我又查看了他们班主任的教学日志，果然是去参观博物馆了。我透过产房的玻璃门看了看老婆，她正在张阿姨的陪同下吃饭，我向她挥手，她和我相视一笑。

突然，手腕上的云终端震了一下，我急忙展开屏幕，公司研发部李总的脸出现在屏幕上。

“小王啊，美国 AndG 公司来了一拨人，他们对我们的产妇预产期预测软件很感兴趣，下午 14:30 你给他们介绍一下，如何？”

“老大，没问题。几号会议室？”

“就安排在 5 号吧，那里有虚拟现实系统。”

现在是 14:00 整，我呼通秘书，请她准备 5 号会议室的会议系统，并开启虚拟现实设备。14:20，我找到一个没人的小房间，把手腕云终端的屏幕调为最大模式，接入 5 号会议室，刚好看到李总带客人鱼贯而入。我开启了家里的电视节目录制程序，15:00—17:00 直播世界杯足球比赛，同时预订 17:30 开始煮饭。我整了整衣冠，然后开启终端上的微型全息摄像头，我“进入”了会议室，“站”在众人的面前，看到他们惊愕的眼神，我自豪地用不太标准的普通话开始介绍。

“各位远道而来的朋友，大家下午好。我是申云公司的王××，是产妇预产期预测软件项目的经理，现在由我来介绍这款软件。”过了一会儿，掌声响起来。

十年前的那次改革打开了中国人勤劳勇敢、创新无限的闸门。此后的十年，中国在经济、军事、科技、教育、国防等领域发生了翻天覆地的变化，很多领域已经超越了美国，每年超过一半的诺贝尔奖都被中国人领走。由此，在世界上掀起了一股学习中文的热潮，在中国留学、就业、定居的外国人一拨接一拨。但凡来中国的外国人，都会一些中文，因此我用中文介绍他们是听得懂的，但是要说得慢一些。

“这软件预测我老婆今天 15:00 临产，所以我上午就到了医院。现在是 14:50，10 分钟后各位就能见证奇迹了。”

“这款软件属于大数据分析软件，中国是大数据非常丰富的国家，横向上有十多亿人口的数据，纵向上记录了每个女人从出生到现在的全部数据，尤其是女人怀孕期间，每隔 1 个小时采集一次数据。在这庞大的数据之上，再利用我们的预测模型，准确率在 90%以上。”又是一阵掌声。

此后发生的事情是这样的：我老婆在 15:20 产下一名女婴，然后张阿姨把一个婴儿微型手腕云终端戴在我女儿的手上，同时向中国国家信息云中心申

请开通了一个账户，输入了虹膜、指纹和 DNA 等信息。中国国家信息云中心下发了身份证号码，并开通了免费的具有 1TB 容量的存储空间。我把照片传到了会议室，并简单介绍了中国国家信息云中心。

“中国国家信息云中心是世界上最大的大数据中心，存储了世界上最庞大的数据，它由中国政府建设和管理，公民免费使用。活着的每个人都有一个账户和 1TB 的存储空间，这 1TB 的存储空间采用分层授权模式，保存着个人生老病死的全部信息，如档案记录、健康数据、活动轨迹、学习工作状况、个人资料等。有了这个大数据平台，我们可以做很多分析和预测。”

“大家应该注意到了戴在我女儿手腕上的像手镯一样的东西，其实绝大多数中国人每人都戴了一个，但是偏远的山区农村还没有普及，那里的网络基础设施正在铺设。这是一个高科技电子产品，功能主要有两个：一是通信终端，二是数据采集器。出生时配发，上小学时更换。使用微波定向技术，所以对佩戴者没有丝毫的辐射影响，孕妇、老人和小孩都可以安全佩戴。它有一个柔性屏幕，完全展开时可达五英寸。它能采集脉搏、心率、体温、体液酸碱度、脑电波、周围环境及位置等信息，采集的信息被实时地传递到中国国家信息云中心。”

……

下午 17:00，我去东方幼儿园接儿子，然后一起回到医院。小家伙见到妈妈就吵着要见妹妹，于是我抱着他通过无菌室的玻璃门往里看，5 号床就是我的女儿，但由于离得太远看不清楚，儿子就说我骗他。

现在中国的妇幼保健系统非常完善，妻子生小孩，丈夫什么都不用操心，生完前三天吃住都在医院，随后医院会把母婴送回家，并配送妇幼用品，以后还会定期赠送产后营养餐、婴儿纸尿裤等用品，而且还安排专门的医护人员，可随时提供咨询和上门服务。

我带儿子在回家的路上去了一趟 3D 打印服务部，根据菜谱建议，我购买了胡萝卜、菠菜和蘑菇，家里还有一些瘦肉，我打算晚上自己炒菜。

一到家，儿子就打开电视，要看 3D 版《大头儿子小头爸爸》。我把电视调到眼睛保护模式，这样儿童即使长时间观看，眼睛晶状体也不会形成习惯性惰性，所以对眼睛没有丝毫影响。

晚饭后，我呼通老家老妈的云终端，儿子也加入了，我发现老妈红光满面，体能状况良好。我顺便说了老婆生孩子的事情，老妈知道现在城市妇幼保健系统完善，根本不用操心，所以她也没有啰唆什么。

儿子去邻居家玩耍了，我开启了虚拟现实设备，重播下午录制的足球比赛节目上半场结束后，比分还是 0∶0。

“王先生，您的银行账户目前有人民币 20.5 万元余额，建议您买进娄胜生物的股票 5 万股，持有时间为三年半，抛掉毛颖地产股，为您女儿购买宝贝人生保险。”壁挂云终端提醒我。我粗略地翻看了一下投资理财云中的信息，同意了这个计划，因为这个投资理财云的算法是我们公司设计的，该算法利用大数据分析，准确率非常高。只不过要成为投资理财云的用户，年费也不低，但公司员工是免费使用的，这是一项福利。

我顺便点开了购物云，了解一下近期家里需要购买什么商品，反馈的主要信息如下：车子需要保养，推荐的保养项目和地点与上次不同，价格降低了；洗发水还能用 5 天，推送购买的是海华波，适合大人使用；牙膏还有两盒，建议购买狮王，经常更换牙膏品牌有利于保护牙齿；父母的老年钙片需要补充两瓶，推送的是最优惠的价格……我看了一下，总价为人民币 5890 元，于是直接点击“同意”按钮，其他的事情全部由购物云搞定，我们等着收货即可。

我通过手腕云终端接入到桌子上的固定云终端上，然后切换到家庭私有云端——壁挂云终端，其实也是家庭的云服务器和 NAS 服务器。在云端，我配置了一套家庭电子图书馆，目前已经收藏了一万多册家人感兴趣的书籍；配置了一套视频系统，收藏了两千多部经典影视作品，并且可以录制节目，可以在电视和各种云终端上点播；配置了一套音乐系统，该系统同时具备卡

拉 OK 的功能，还能根据人的情绪变化播放合适的音乐；配置了一套游戏系统，安装的都是一些开发智力的游戏；配置了一套家庭监控系统。同时，卧室、书房、阳台、客厅各摆放了一台固定云终端，终端屏幕采用薄如纸张的钢化玻璃，会自动感应周围的接入设备。当我靠近它时。固定云终端自动接管手腕云终端的屏幕，每台固定云终端都是通过手腕云终端做接入认证的。

目前，我有三个云桌面：一个是国家免费提供的中国国家信息云中心的公有云桌面，一个是公司的私有办公云桌面，还有一个是家庭私有云桌面。不过，我另外购买了下面的云应用：文字材料处理、实时天气预报、开发测试云、学习云、出行云、购物云等，这些云应用已经集成到我的公有云桌面上了。我个人的文档资料统一存放在中国国家信息云中心，这样无论何时何地，我都能访问自己的资料，就算云终端丢了，个人信息也永远不会丢。我还在家里组建了物联网，把冰箱、电视、空调、洗衣机、电饭煲、微波炉、护理机器人、监控设备等全部通过无线或射频接入家庭私有云终端，从而可以远程控制这些家电设备。

晚上睡觉前，我接通了老婆的云终端，看到她睡得很好；我又呼通女儿的云终端，发现护士正在给她换尿布。我们一家人的云终端都默认设置了互信模式，只要本人没有按下“隐私”按钮，我们随时都可访问对方的云终端，并控制对方的微型摄像机，查看周围的环境。

第二天，送走儿子后，我早早来到公司。虽然我是公司的二级荣誉员工，上下班不用考勤，没有固定的任务安排，但我喜欢公司的工作环境，帅哥美女一大堆，尤其是坐我左侧的尤韵丽，简直就是仙女下凡，智商高得让帅哥们望而生畏。奇怪的是，这样的美女竟然喜欢哲学，动不动就说“事物的本质是什么”。我不赞同她主张的“哲学就是元知识”的观点，信仰才是知识的原点。记得那次入职教育，我和新员工分享了人类知识架构模型，尤韵丽眨巴着大眼睛，一脸迷糊，后来她几次找我探讨，才算是对我信服了。

我阐述的人类知识架构模型是这样的：信仰是原点，之上是哲学，然后依次是艺术、人文、社科、科学、技术和工具。其中，底三层代表人类的文

明，也是人的精神、思想和心灵的载体，更是人之所以与动物不同的原因。科学就是发现，技术就是模仿，今天的计算机与猿用来掏洞里白蚁的木棍没有本质的不同。

我和同事们相处得非常融洽，和他们坐在一起办公，心情特别愉快。另外，公司的桌面固定云终端配备弧形屏幕和视距变换模块，长时间对着屏幕办公，对眼睛没有丝毫影响。这个云终端是我们公司开发的，销售到世界各地。在去年年会上，记得公司总裁提到过，已卖出 100 万台，由此申云公司成为世界上最大的云计算公司。

我来到 19 楼的研发三部，门禁系统马上和我手腕上的云终端建立联系并验证了我的身份。在我走到离门 1 米的地方，门自动开了，我几乎没有停留地直接走到我的办公桌边。手腕在固定云终端左侧一晃，马上就进入了我的私有办公云桌面。昨天未完成的方案文档仍然处于编辑状态，同时屏幕上跳出了一个窗口，是研发三部李总代表大家发来的贺电，上面还有我女儿的照片，末尾是“今晚大家去龙泉食府探讨一下移动虚拟现实技术”。又来这一套，看来我又得破财了。研发三部是公司的要害部门，我们的云桌面处于公司云中的二级安全环中，只有一级安全环和授权的其他二级安全环中的部门才可以访问，而且采用了基于虚拟机的独立云桌面，隔离效果非常好。

突然屏幕上跳出窗口，是公司研发部郁副总呼我，我单击“同意”按钮，郁总的头像就出现了：“小王，那个远程手术网络延时问题有没有突破？我们的合作方等不及了。”

“郁总，我可以肯定，延时只跟网络路径上的转发节点的性能和数量有关。”我顺便把我的计算机桌面投射到郁总的云终端上，并调出白板，“也就是说，跟广域网的架构相关。理论上，最小延时就是最短路径上的延时，在网络路径两端投入更多的精力只能增加带宽，而对减少延时毫无作用。”

“你有没有解决办法？”

“办法只有一个，那就是减少路径上的转发节点数目，控制在 4 个节点以

内，回路延时就能控制在 100 毫秒以内，这对于远程手术来说是可以接受的。”

“说说具体的办法。”

“一是建议国家把远程医疗纳入快速网络系统中；二是改变病人或者医生的地理位置，比如省医院的医生只能远程做地市级医院的手术，地市级医院的医生只能远程做县级医院的手术。如果某个乡的患者要接受省医院医生的远程手术，那么他必须到市级医院去。不过，我们应该对整个江苏省的网络延时做一下实测和统计。”

“这个好办，我把小廖呼进来。”小廖是公司售前工程师，最近几个月都在江苏出差。一会儿，小廖的头像就出现在屏幕上，同时展开的还有他手腕云终端上的小屏幕。郁总说：“小廖，你这周把江苏从省医院到各个地市级医院、县人民医院、乡医院的网络延时统计出来。另外，地市级医院到县人民医院、乡医院，县人民医院到乡医院的数据也要。”

……

后来，我又跟李总讨论了一个上午。中午研发部的同事一起去公司的 6 号餐厅吃饭，那里配备了几十台 3D 食品打印机。大家依次用手腕云终端在打印机右侧一晃，然后找个大圆桌坐下，天南海北地聊着。聊着聊着，尤韵丽又扯到哲学上去了。

“你们说苏格拉底主张的一个城邦里的男女同吃同住、共妻共子，好不好呀？”

“我看行，哈哈。”

“我有一个问题，如何解决近亲‘结婚’生子的问题？每个人都知道自己的母亲，但是不知道自己的父亲是谁。”

餐厅机器人陆续把定制的午餐送过来了，大家一边吃一边继续聊。

“也不知道自己的母亲，因为一生下来就被集中抚养和哺乳，对于一个城邦来说，同一天出生的人太多了。”尤韵丽补充说。

“要不‘结婚’前先做个滴血认亲？”

“这是迷信，根本不准。不过，可以规定只能与城邦外的人结婚。”

“这存在一个男人与他的外甥女结婚生子的可能。”我提出疑问。

“刚才阿毛提到的滴血认亲虽然迷信，但倒是提醒了我，我们自己来研发手持 DNA 检测仪，通过两个人的头发立即检测出 DNA 相似度，或者干脆直接显示结果，是否允许这两个人‘结婚’生子。”尤韵丽显得有点兴奋。

“还研究啥，现在每个人的 DNA 都已经登记在国家监护中心了，只要法律允许，我们随时都可以查到。”

“啊？是啊，我们都还以为生活在苏格拉底时代呢，哈哈。”

……讨论没有结果，大家回到各自的休息室午睡去了。但我毫无睡意，看了一眼妻子、女儿和儿子，然后呼通龙泉食府订餐终端，预订了那个最大的孔乙己包房，并把订餐信息定时到 15:00 发到研发三部全体同事的终端上。最后又想起了那该死的网络延时问题。

下午 14:20，我走进了远程教学中心的 6 号格子间，14:30，我要给广西西林县 25 所三年级小学生上公民教育课。西林县是最早搞教育云试点的县之一，也是由我们公司免费承建并开展远程教学任务的，现在已经对接到全省的教育云之中。我们为每个课桌配备了超轻超薄的云终端（都是用高强度钢化玻璃做成的，用锤子也砸不烂），同时用巨大的弧形多点触摸屏代替传统的黑板，并且安装了虚拟现实系统。其中，公民教育、自然及科学三门课的教学任务一直由我们公司负责，其中研发三部负责公民教育课，客服部负责自然课，科学课没有对口部门，由员工自愿报名，人事部门考核通过后方可上课。

我开启了虚拟现实系统，唉，这些小家伙又是这样，简直要拆房子了，我“啪啪”地猛敲课桌，同时把系统切换到远程教学状态。学生们看到“我”出现在讲台上，马上坐回到自己的座位上，但并没有马上安静下来。

“对于木沉村小学的马云彪同学现在的做法，其他同学说说他做得对不对

啊？”

“不对！”

“为什么不对呀？”

“上课了，他还在大喊大叫。”

“他影响别人听课。”

“……”

“同学们说得很对，上课了如果还大喊大叫，就会影响其他同学听课。”由于大家都在看马云彪同学，他红着脸很快便安静下来。我得表扬他一下。

“不过，马云彪同学也有做得好的地方，他不怕老师，敢于发表自己的观点，大家为他鼓鼓掌吧。”

啪啪啪……课后，我把下面的作业题发送到每个学生的云终端里：

“找出父母的一个你认为错误的观点，并给出理由。”

离开了可爱的孩子们，我去了医院，顺路捎上儿子。女儿躺在妻子的怀中，老婆正在和岳母视频通话，我们都凑过去，让岳母看看我们一家四口的模样。

下午 18:00，我带着儿子到了餐厅，同事们差不多都到齐了。但是吴新华没来，这家伙自从生了个儿子后，下班就回家陪儿子去了。落座后，我把女儿的视频投放到墙上，让各位“膜拜”了一下。

“来，大家一起敬王工，恭喜他喜得千金。”李总首先发话了。

“恭喜，一起喝掉！”

“你们说说，废除计划生育政策后，也没见大家都生一窝仔啊，这是为什么？”

“我不知道原因，但我知道生两个就足够了。我喜欢和大家一起工作、一

起喝酒，闲时带上家人一起外出旅游。”我说。

“你说到根儿上了。现在社会福利好，工资又高，没有压力，人人都喜欢工作、旅游、一起喝酒，享受生活。”

“对，以前是为了下一代的幸福辛苦自己，但结果是代代辛苦；现在是享受生活幸福自己，结果是代代幸福。其实本质的原因是人们所处的环境变了，变得和谐、美丽、没有压力，人人心中充满喜乐。”尤博士又在说本质了。

“现在的机器人基本上代替了人类的体力劳动，人类的工作全是创造性的工作及信仰、哲学、艺术和人文社科方面的了。你们看到新闻没？今天早上的，说那个人工智能云已经输入了全人类 75%的既有知识了，基于智能云的机器人终端具备相当的研究创造能力。自从阿法狗完胜李世石后，人类再也不好意思举办国际围棋赛了。未来我们都不用动脑、动手啦。”大家喝着酒、吃着菜、聊着天，慢慢转到了正题上。

“今天下午上课，有一个学生趴在桌子上，我调看了他的体能数据，显示发烧。我很想走过去抚摸他的头以示安慰，但我远在千里之外啊。”

“好，那我们接下来讨论讨论如何实现移动的虚拟现实技术，让老师可以在教室里走动，使同学们能感觉到老师在向他走来，抚慰他、关爱他。”李总开启了话题。

“我们现在的技术还处于初级阶段，只实现了视觉和听觉的部分功能，触觉、嗅觉、味觉几乎是空白。”

“所以离真正的能混淆现实和虚拟界限的虚拟现实系统还差很远，这涉及的技术太多了，还有跨学科的内容。”

“云计算能提供无限的计算能力，网络带宽也不存在问题，现在的关键点是感官终端设备，要实现真正的全浸入式虚拟现实系统，离不开手套、头盔、衣服、鞋子这些终端设备。”

“我们公司作为世界上最大的云计算公司，理应要在这些方面有所贡献才

对。”李总说。

……

讨论的结果是李总向上级申请一个研发项目，项目取名为“触觉感知”。

……

不知不觉过了三个月，老婆和女儿已经从医院回到了家里，儿子开始读小学了，他的手腕终端已经更换。看到他们快乐地生活、学习、成长，作为一个中国人，我感到无比自豪。

“触觉感知”项目早已批下来了，而且得到了政府的资助，不过我不是项目组成员。

下午 15:00，工程四部一群云计算工程师坐上了公司的商务机飞赴贵州仁怀市，那里的名酒工业园区联合云是我们公司几年前实施的云计算项目。这次应甲方的要求实施云端服务器扩容，云服务器已经先行送达。今日的行程安排本来没有我，但我想从用户的角度去看看云计算通信协议的使用效果。我依然记得五年前参与投标的情景，标书还是我起草的，取了一个豪迈的题目——《以云计算实现酿酒产业链全控制・助力仁怀市向“国酒文化之都”迈进》，下面还有一个小标题“——针对遵义市仁怀名酒工业园区的云计算解决方案”。后来参与吉林通化医药高新技术产业开发区的联合云项目招标，我继续把标书写得很豪迈——《以云计算提升园区软实力・助力园区向“国际化医药名城”迈进》，再加一个小标题“——针对通化医药高新技术产业开发区的云计算解决方案”，招标的结果可想而知。

飞机在蓝天白云中飞翔，我透过地板观景窗鸟瞰祖国的大地。看啊，那是绿意盎然的太原，新能源汽车取代了煤炭成为这里的支柱产业；那是黄河，像一根彩带，水不再那么浑浊；那是高原，黄色不再那么抢眼；进入了河南，一马平川，秋收在即，又是一个丰收年；哦，看到了长江，它恢复了千年前的气势，奔腾呼啸；飞过重庆上空，一个花园般的城市展现在眼前……祖国，看到伤疤慢慢愈合、体格逐渐强壮的您，我感慨万千。我下意识地握紧了拳

头，为了这个国家，我必须奉献自己！

飞机开始向下滑翔，迎接我们的首先是时隐时现的酱香型酒味。“嘟嘟”，领队的手腕云终端响起来，他展开一看，咧嘴笑了：“马总发来了迎接词呢，说‘兄弟们，今晚酒里泡泡，一醉方休’。”酒乡就是酒乡，直接泡酒。

我们一行人照例先来到云端机房，那栋三层楼高的造型很别致的灰色建筑，半米厚的钢筋混凝土墙壁表面覆盖着从意大利进口的海底岩石板，这种石头具备超强的信号屏蔽效果，手机在里面根本没有信号。整栋楼里面就一层，一进门首先是更衣室、防火墙，上方是一个凸台，凸台上设立着摄像监控设备和巡视岗位。往里就是阶梯式的机房了，阶梯上摆放着机柜，阶梯间采用具备防火效果的加厚高透明钢化玻璃分隔，层级越高，安全级别越高。我们依次穿过 5 级门禁，隔着玻璃看到了最高层级的三台服务器闪着一排幽蓝色的小灯。在我看来，这就是云端的心脏，欢快闪动着的蓝色小灯告诉我们它的心脏是多么强壮。在我们脚下半米厚的钢筋混凝土下面是供电系统，采用三级多路供电机制，确保云端供电六个“9”的高可用性。云端采用水冷机制，里面的温度恒定在 25℃，流出的水温达到 50℃，热水集中到一个保温水池里，然后通过热水管流入各家各户。如果把整栋楼看成一个黑盒子，那么流进去的是冷水和电力，流出来的是热水和强大的计算能力，无谓损耗的能源非常小。

在这个云计算中心“漂浮”着这样几朵“云”：企业应用云（如 ERP、CRM、SCM 等）、电子商务云、电子政务云、教育培训云、移动办公云、数据存储云、物联网云、高性能计算云，覆盖了从田间到餐桌的酿酒全产业链，打通了上下游企业的信息通路，避免了恶性竞争，使得“贵州白酒”品牌和“贵州茅台”品牌价值不断攀升。

第二天一大早，我驱车沿着赤水河北上，行驶在两侧大树覆盖的青色马路上。我索性打开天窗，早上清新的夹杂着酒香的空气从头顶灌下，使我如痴如醉。昨晚小麦呼我，说她的孩子明天刚好九岁，他们都想见我。我看了一眼旁边的蛋糕、玩具枪和儿童玩伴机器人（都是昨晚打印出来的），九年前

的那个晚上，那惊心动魄的一幕马上浮现在我的脑海里。沿路所见使我惊叹于勤劳的中国农民的智慧，只要不阻止他们劳动和破坏他们的财富，他们就能创造出一个美好的家园。加上政府对农民的大力扶持，这才五年的时间，这里家家别墅、人人汽车、村村马路、处处酒香，教育、医疗、养老全免费了。

上午 8:30，我看见了被绿树环绕的河腰坎村，几年不见，这里变化好大。但那村头朝我挥手的小麦却依然风姿绰约、楚楚动人，我老远就认出她来了。她牵着的男孩肯定就是麦小聪。

我在远程医疗室门口停车，小聪扛着玩具枪跑开了，这小家伙肯定是找其他伙伴们玩耍去了。

他爸是英雄，在那边；他妈是英雄，在这边，他和她都是北京医科大学的研究生，郎才女貌，他们两个人的命运在那年发生巨变，之后她怀着身孕一个人回到老家。那年的 11 月，我出差到这里，主要任务是摸清光缆损毁情况，晚上就住老乡家。夜里一个老婆婆的一声撕心裂肺的呼叫惊动了整个村子：小麦难产，母婴危在旦夕！“英雄必须留后”，我一跃而起，顺手取走卫星可视电话，直接冲入人声鼎沸的房子。男人们在外面急得团团转，里面传来女人的哭声和接生婆的哭腔：“咋办咧？咋办咧？”

“别慌，我要进来了！”等了十几秒后，我进入了房间，发现小麦还能说话，于是让她赶紧打电话给她的导师。小麦的导师是全国有名的妇产科专家，电话打通了，导师稳重自信的声音使得我们很快安静下来，我开启视频并且和导师的计算机连上。我双手举着卫星电话并做翻译，由接生婆来操作，一切都有条不紊地进行着。我生平第一次见证了生命的伟大诞生过程，“母爱”、“生命”，字字珠玑，金山银山在这里显得轻如鸿毛，争权夺利在这里显得幼稚可笑。

最终母婴平安，全村释然，而我成了麦小聪的干爹。后来在她导师的帮助下，我带领公司团队完成了贵州省首个远程医疗室的建设，小麦成了这里

有名的医生。现在每个村子都建设有相同的医疗室，而且全部接入了全国医疗云平台。小麦为十里八乡做了不计其数的远程手术，而且她的卫生保健云网站开展得有声有色，注册用户过十万，有近五千名用户购买了小麦的卫生保健咨询服务。小麦服务好、收费低，大部分保健品直接取自这里的青山绿水间，纯天然、无污染。

我们走进医疗室，房间内一尘不染，这里的设备先进，有虚拟现实系统、远程手术刀、远程腹腔镜等，几个穿白大褂的阿姨都在忙于收集各种器材。

“她们都是来自附近乡村的志愿者，还有几个大学生志愿者今天没来。”小麦一边和她们打招呼一边介绍，看到她们“诧异”的眼神，我就知道她们在想什么。

“小麦，这么多年了，你还放不下他吗？”

“他是一个英雄，早已经融入了中国欣欣向荣的每个领域中、每个城市中、每个村落中，我放不下他，他们也忘不了他。”望着她自豪的、发自内心的一张笑脸，我转变了话题。

接下来的两天，我整理了远程医疗室中这几年的云计算通信日志，并且由小聪带路，跑遍了这里的山山水水，测试了每个角落的信号强度、带宽和延时。在云中的地图上标记了这里每处险要的地方，并写下注意事项，最后把地图发给小麦，并叮嘱她采药时给我开启位置信息。

……

第 2 章

“云”概念

云计算在眼下的中国呈现出冰火两重天的怪象：这厢，云服务提供商们个个摩拳擦掌、热情高涨，大家恨不得从“万亿云计算市场”蛋糕中分得一大块，却鲜有人脚踏实地做产品；那厢，用户们迷茫、观望者甚多，大家纷纷捂紧各自的钱袋，弱弱地问：“云计算到底是什么东西？能给我带来什么好处？”国内，大手笔的云计算中心像雨后春笋般在神州大地上拔地而起，只见机房机器轰鸣，壁挂大屏幕闪烁，却不见云应用，这等同于天上电闪雷鸣，就是不见下雨。深圳超算中心，强大的计算能力没有得到充分利用，能源无谓消耗严重。

“云计算”可谓是目前大家理解得最为混乱的概念之一了，要么是学术味特浓的定义，让人看了不知所云；要么是举例式描述，没有抓住本质，而且将其比喻成“水”、“电”、“煤气”等，很容易误导读者。诚然，“云计算”是一个技术性很强的概念，要想给普通读者解释清楚确非易事。尽管如此，把“云计算”说清楚是很有必要的，因为云计算正扑面而来，绝大多数人已经在

使用它并从中获益。可以预计，再过五年，我们每个人的生活都将被云计算逐渐改变，就像之前的互联网一样，只是云计算对人们生活的影响将更深刻、更持久、更彻底。

云计算不是新的技术，而是传统 IT 技术的集大成者，因此要想讲清楚它的概念，必须得从 IT 的传统技术讲起。

2.1 软件的概念

软件，也叫程序，那么究竟什么是软件？在回答这个问题之前，先来看下面的例子。

老李要求他的痴呆儿子（下称小李）在一张纸上画三角形并计算出三角形的面积。老李准备了两张白纸：一张白纸上标注了三个点：*A* 点、*B* 点和 *C* 点，以及每个点的坐标；另一张白纸上写下了画三角形的步骤和计算三角形面积的方法。小李只要按照这些步骤和计算方法动手即可。步骤如下：

（1）画一条直线连接 *A* 点和 *B* 点。

（2）画一条直线连接 *A* 点和 *C* 点。

（3）画一条直线连接 *B* 点和 *C* 点。

（4）根据下面的公式计算三角形的面积并写在纸上：

$$(x_a y_b + x_b y_c + x_c y_a - x_a y_c - x_b y_a - x_c y_b) \div 2$$

（5）把那张画了三角形并标注了面积的纸张交回给老李。

老李把这两张纸交给小李后，小李在书桌上摊开这两张纸开始工作。大概过了十分钟，老李得到了一张画好三角形并标注了面积的纸张，任务完成，如图 2-1 所示。

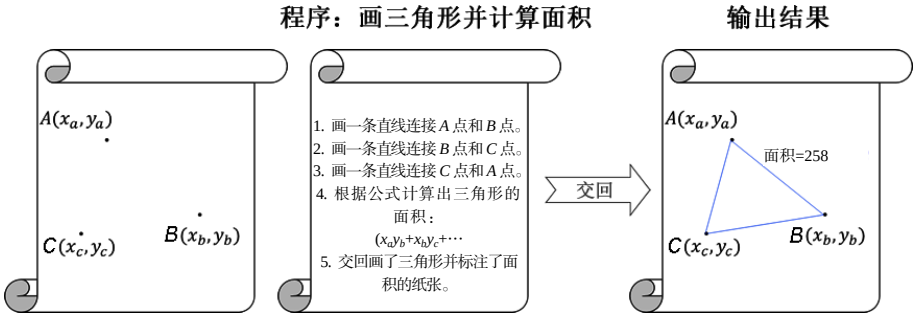


图 2-1　画三角形并计算面积（1）

现在把小李比喻成计算机的 CPU（中央处理器），绘制三角形的步骤就是软件，一个步骤就是一条指令（语句），老李就相当于程序员，而那张画了三角形并标注了面积的纸张就相当于软件的输出（在屏幕上显示或者打印出来），“画一个三角形并计算面积”就是软件要完成的任务（程序要处理的数据）。

几天后，老李要画更多形状不同的三角形并计算出各自的面积，如果还是按照原来的方式，那么针对每个三角形，老李都要给小李提交两张纸，一张纸标明三个点，另一张纸写明步骤。这样做一方面浪费了纸张，另一方面增加了工作量。于是老李想出了一个办法，只交给小李一张写有步骤的纸张，画一个三角形并计算面积的步骤改为：

（1）询问 A 点的坐标。

（2）询问 B 点的坐标。

（3）询问 C 点的坐标。

（4）画一条直线连接 A 点和 B 点。

（5）画一条直线连接 A 点和 C 点。

（6）画一条直线连接 B 点和 C 点。

（7）根据下面的公式计算三角形的面积并写在纸上：

$$(x_a y_b+x_b y_c+x_c y_a-x_a y_c-x_b y_a-x_c y_b) \div 2$$

（8）把那张画了三角形并标注了面积的纸张交回给老李。

这样小李收到老李递过来的纸张后，先询问老李三角形的三个顶点的坐标，然后再去画三角形并计算面积，如图 2-2 所示。

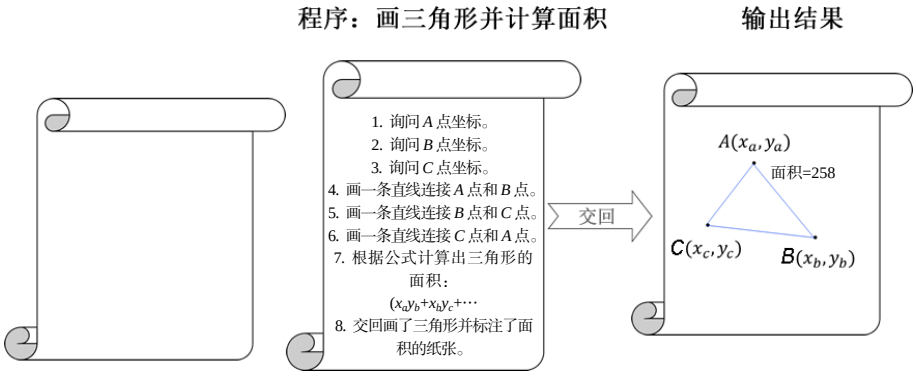


图 2-2　画三角形并计算面积（2）

现在不只是老李，任何人只要复印那张写了步骤的纸张，都可以叫小李画出任何他们想要的三角形，只不过小李在画三角形前要询问顶点的坐标。继续把这些步骤比喻成软件，那么相对于前面的软件，这次增加了三条输入语句，用户在运行这个软件时，要用键盘输入三角形的三个顶点的坐标，然后就会在屏幕上看到一个三角形及这个三角形的面积。

至此，我们总结出这个软件已经具备如下特征。

- 完成任务：画三角形并计算面积。
- 输入数据：顶点坐标。
- 输出结果：三角形及三角形的面积。
- 指令集：详细定义画一个三角形的步骤和计算三角形面积的方法。

指令集保存在一个文件中，这个文件就叫可执行程序，允许存放在硬盘、U 盘、光盘或者网盘中，可以任意复制和传播。比如，Windows 操作系统中的“计算器”这个程序(指令集)保存在硬盘上的 C:\Windows\ system32\calc.exe 文件中，用鼠标双击它或者单击“开始”→“所有程序”→“附件”→“计算器”，即表示命令 CPU 现在就按照里面的步骤进行操作（专业术语叫执行

指令）。

最后，我们通俗地定义一下软件：由程序员写的需要让 CPU 来完成某项任务的步骤。只不过这些步骤是用计算机语言来描述的。常见的计算机语言有 C 语言、C++、Java、PHP、Go 等，编程人员必须严格按照计算机语言的语法规则来写程序，如下面用 C 语言语句实现的加法运算。

```
int x, y, sum;                  ←定义三个容器 x、y、sum，用于存放两个加数及其之和
loop:
scanf( “%d %d”,&x,&y);          ←等待用户用键盘输入两个加数
sum=x+y;                        ←求和
printf( “和是：%d\n”,sum);      ←在屏幕上显示加法的和
goto loop;                      ←继续下一次加法运算
```

上面左侧灰色框内就是程序员用 C 语言写的两个数相加的程序语句，右侧是说明信息。

软件必须包含输入/输出语句和计算语句，没有包含输入/输出语句的软件没有任何用途，因为它就像一个黑盒子，既不能输入任何东西，也不能从它那里得到任何东西。在这里，我们要澄清两个概念：实时输入/输出和批量输入/输出。实时输入/输出是指 CPU 执行输入/输出步骤时，立即完成输入/输出动作；而批量输入/输出是一次性输入全部的信息，一次性输出全部的计算结果。大部分计算机软件，如办公软件，都要求实时输入/输出；绝大部分网站都是批量输入/输出的，比如注册一个在线免费邮箱，我们要一次性输入全部的注册信息，然后再单击“提交”按钮，送出全部的输入信息。

实时输入/输出软件可进一步划分为强交互性软件和弱交互性软件两种，强交互性软件是指在运行时需要实时地进行大量输入/输出操作，且输入之后马上能看到输出结果；而弱交互性软件是指软件运行时实时地进行少量的输入操作，然后就源源不断地输出了。属于强交互性软件的有：微软的办公软件（Word、Excel、PowerPoint）、记事本、QQ、Photoshop、AutoCAD，金山的办公软件 WPS、金山词霸，Visual Studio、Eclipse、Vim 等，共同特点是用

键盘输入的内容马上能在屏幕上显示出来；而下面这些软件属于弱交互性软件：酷狗音乐播放器、Adobe Reader、家庭相册、迅雷下载等，共同特点是在程序运行初期只需输入少量信息，然后就源源不断地输出了。

在规划云计算方案时，要特别关注软件的输入/输出是实时的还是批量的。如果是实时的，那么还要进一步区分是强交互性的还是弱交互性的。对于实时的强交互性软件，有两种解决方法：（1）计算机网络的延时控制在合理的范围内（一般要小于 100 毫秒），解决方法是就近部署云计算分支中心。如果延时过大，那么当使用诸如 Word 等办公软件时，需要等一会儿才能在屏幕上看到刚才用键盘输入的字符，用户体验很差。（2）改造软件以便能通过网页浏览器访问，用户只与本地的网页浏览器进行实时输入/输出交互，而网页浏览器与“云”中软件进行批量输入/输出传输，使用浏览器改造强交互性软件如图 2-3 所示。

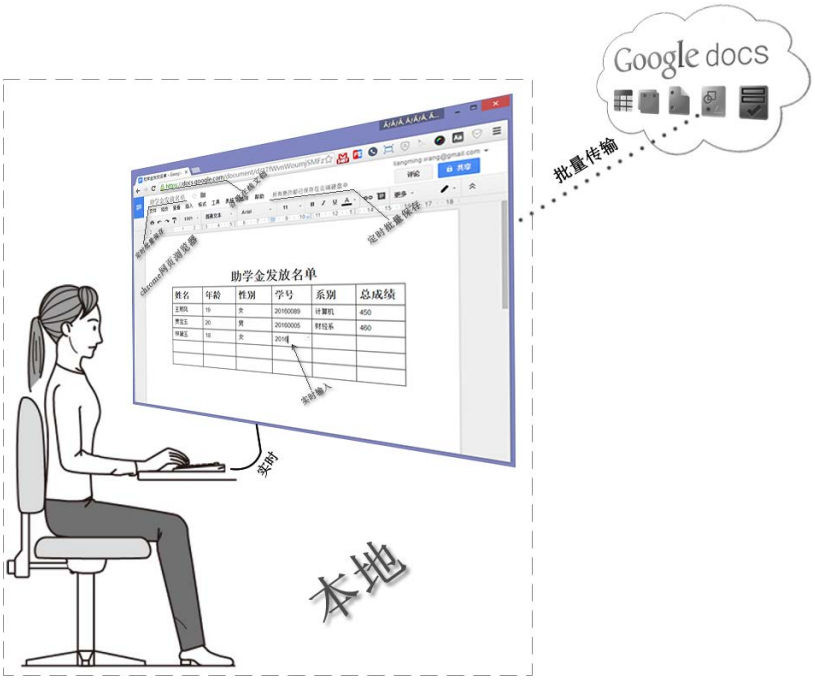


图 2-3　使用浏览器改造强交互性软件

计算机网络延时的概念在 2.3 节会详细介绍。

最后，我们用图 2-4 来表示一个软件运行模型。

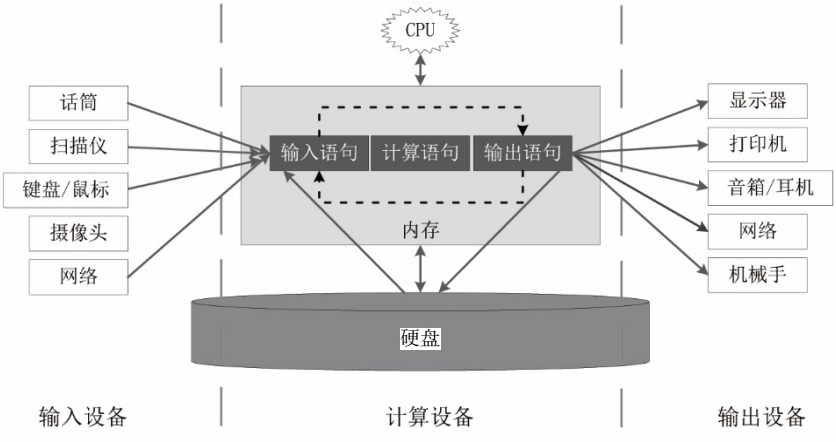


图 2-4　软件运行模型

一个软件以文件的形式保存在硬盘上，当我们用鼠标双击它时，这个软件就被读取到内存中，此后 CPU 就按照里面的步骤一步步执行。执行到一些输入步骤时，就要从输入设备上获取信息（常见的输入设备有键盘、鼠标、扫描仪、话筒、摄像头、网络等）；执行一些计算步骤时，要用到计算设备；执行输出步骤时，把计算的结果通过输出设备输出（常见的输出设备有计算机显示器、打印机、绘图仪、音箱、耳机、网络、机械手等）。对于一些较大的软件，不一定就是按照“输入—计算—输出”的顺序进行的，在软件执行的过程中，随时可能需要输入，也随时可能会输出，计算步骤也可能安排在任意时刻。

注意：硬盘上的文件既可以作为输入设备，也可以作为输出设备。例如，编辑一个已经存在的 PPT 文档，首先作为输入设备，PPT 文档中的内容被读取到内存，编辑完成后保存时又作为输出设备，内存中被修改的内容又被写入这个 PPT 文档中。计算设备一般指 CPU、内存、存储器（硬盘属于最典型的存储设备）和网络，为什么网络也算计算设备呢？在云端运行一个分布式

应用程序时，网络是必需的。换个角度来看，计算设备就是程序运行时需要使用的资源——计算资源。硬盘上的 PPT 文档本身不是程序，只是用于输入/输出的数据文件，双击它能打开进行编辑，实际上运行了微软的办公软件中的 powerpoint.exe 程序，因为在安装办公软件时自动建立了数据文件和程序的关联，建立好关联之后，只要双击数据文件，就能运行关联的程序。

最后我们总结一下本节的知识点：

（1）软件是由程序员写的让 CPU 来完成某项任务的步骤。

（2）这些步骤包括输入/输出步骤和计算步骤两大类。

（3）输入/输出步骤要使用输入/输出设备，计算步骤要使用计算设备，计算设备也称为计算资源。

（4）键盘和鼠标是最常见的输入设备，显示器和音箱是最常见的输出设备，CPU、内存、存储器和网络统称为计算资源。

（5）软件平时保存在硬盘中，但必须读取到内存中去执行，一个软件可以被多次执行。

计算设备和输入/输出设备的分离是云计算的特征之一。也就是说，对于云计算而言，计算设备位于远方的云端，而输入/输出设备就在眼前。究竟如何分离计算设备和输入/输出设备，请看后续章节。

2.2 计算机系统

组装一台计算机的步骤大概就是写好计算机配单、采购零部件、组装、安装操作系统和各种应用、交付给用户使用，更详细的描述如下。

（1）写好配单，重点考虑用户的需要和各种零配件之间的兼容性。

（2）买齐配件，包括 CPU、主板、内存、电源、硬盘、机箱、显示器、键盘、鼠标、音箱等。

（3）把配件组装到一起，此时得到一台纯硬件的裸机。

（4）安装操作系统，如 Windows 10。

（5）安装驱动软件，此时得到一台只安装了操作系统的计算机，这里称为平台机。

（6）安装需要的应用软件，如聊天软件、办公软件、音/视频播放软件、上网软件和游戏软件等，此时得到安装了操作系统和应用软件的准计算机系统。

（7）最后把以前备份的数据资料（如文档、照片、视频等）复制到计算机硬盘中，这样完整的计算机系统就诞生了。

由此可知，完整的计算机系统包括硬件、软件和数据资料。软件又可分为平台软件和应用软件，操作系统和数据库软件是典型的平台软件。应用软件种类繁多，涉及人们生活的方方面面，如聊天软件、办公软件、上网软件、音/视频播放软件、图像处理软件等。计算机系统体现出如图 2-5 所示的逻辑层次结构。

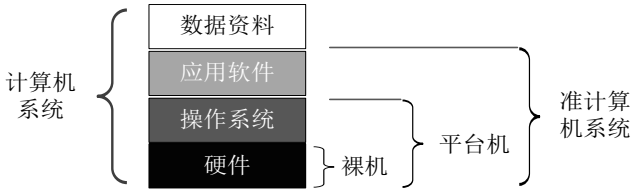


图 2-5　计算机系统逻辑层次结构

传统的个人计算机由于操作系统没有固化，感染病毒、不正常关机、误删除重要文件、配置不正确等都可能导致操作系统损坏，由此导致死机、蓝屏等各种问题。这些问题一直困扰着计算机用户，而重装操作系统、应用软件并恢复数据资料要消耗大量的时间和精力，严重时甚至丢失数据资料或者

外泄私人资料。所以说，传统的计算机系统是专家系统，意思是说只有计算机专家才能很好地使用计算机，普通民众使用计算机会面临很多困难。

现在有两个趋势：第一个趋势是操作系统固化，用户自己不能安装和更改操作系统，只能在线对操作系统进行升级。应用软件也不能随便安装，只能从官方应用软件库中在线安装，如平板电脑、智能手机等。第二个趋势是云计算，后面的章节会展开讨论。

人们为什么要购买和使用计算机？也就是说人们购买和使用计算机的目的是什么？目的只有一个，那就是处理数据资料，如上网查看网站信息、编辑 PPT、写个人简历、编辑图片、听音乐、看电影、和他人传递聊天信息（聊天）、发微博等，其实这些只是一些数据资料的浏览、编辑、传递和存储罢了。如果没有裸机，或者有了裸机但没有安装操作系统，或者安装了操作系统但没有安装应用软件，那么就无法达到处理数据资料的目的。由此可以说，准计算机系统（硬件、操作系统、应用软件）是手段，数据资料是目的。不难想象，如果没有数据资料要处理，就根本没必要购买和使用计算机。

手段与目的分离是云计算的另一个特征，即云计算服务提供商拥有计算资源这个“手段”，而云计算用户拥有数据资料这个“目的”。

2.3 计算机网络

今天是周末，位于深圳南山区荔湾小区 28 栋 801 房间的云计算专家正在通过计算机和北京海淀区学院路文山花园 30 栋 2802 房间的做服装生意的朋友解释什么叫“云计算”，双方你一句我一句地在计算机上互传信息。做服装生意的朋友越听越糊涂，最后云计算专家输入了如下一句话：

“如果我能给你解释清楚，那么我的智商就下降到和你一样的水平了。”

然后单击“发送”按钮，对方马上反馈了一个晕倒的表情，聊天就这样无疾而终。现在的问题是，单击“发送”按钮后，“如果我能给你解释清楚，

那么我的智商就下降到和你一样的水平了。”这句话如何能准确无误地立即显示在北京海淀区学院路文山花园 30 栋 2802 房间里的那台计算机屏幕上呢？而地球上千千万万的其他计算机就不会显示这句话呢？这两个问题可归结为：如何把一台计算机发出的信息准确无误地发送到另外一台计算机上？这个问题就是计算机网络所要解决的问题，两台计算机进行通信如图 2-6 所示。

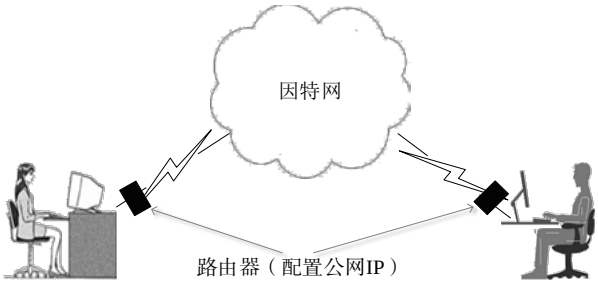


图 2-6　两台计算机进行通信

写过纸质信件的人很清楚，为了能使信件送到对方手中，必须要在信封上写清收信人的详细地址和姓名，且“地址+姓名”是唯一的。至于这封信具体如何传递，那就是邮局的事了，不用我们操心。如果两台计算机之间要通信，那么同样每台计算机要定义一个唯一的地址——32 个由 0 和 1 组成的二进制数字（为了便于人们记忆，常把 32 位 0 和 1 组成的数字分成四段，每段 8 位，8 位二进制数再换算成十进制数，最后写成 xxx.xxx.xxx.xxx 格式，如 192.168.0.10 对应的二进制分段为 11000000.10101000.00000000.00001010，因此最终的 32 位二进制地址是 11000000101010000000000000001010），有点类似于身份证号码，且必须全球唯一，这不同于邮局寄信的“XX 市 XX 区 XX 街 XX 号”格式的地址，计算机的地址称为 IP 地址，格式是“xxx.xxx.xxx.xxx”。有读者可能马上会想到这样一个问题：32 位二进制数字最多能给多少台计算机分配唯一的 IP 地址呢？答案是 2^{32}，约等于 43 亿台计算机。目前 32 位的地址（简称为 IPv4 地址）早已分配完毕，其中分配给中国的 IPv4 地址非常少，只与分配给微软一家企业的数量相当。在中国，公网 IPv4 极度匮乏，因此租用一个公网 IP 地址，价格非常昂贵。为了解决 IPv4 地址不够用的问题，人们

发明了 IPv6。IPv6 采用 128 位二进制数字编码，可以分配的地址数量可达 2^{128}，这是一个天文数字，平摊下来，地球上每平方米可以分配上百万个 IPv6 地址。但是现在 IPv6 地址还没有被普遍采用，尽管现在的网络设备都已支持 IPv6。聪明的中国人还发明了动态域名服务（DDNS，比较有名的产品有花生壳等），利用动态域名服务，解决了外网如何通过域名访问局域网内的计算机的问题。

现在我们再来看看深圳的云计算专家发送的“如果我能给你解释清楚，那么我的智商就下降到和你一样的水平了。”消息是如何传递到北京朋友面前的计算机的。单击“发送”按钮后，消息、自己计算机的 IP 地址、对方计算机的 IP 地址被打包在一起并通过宽带发送给深圳电信，然后电信内部传递，最后到达北京海淀区电信后，由海淀区电信通过对方的宽带发送给对方计算机。注意：消息的打包和传递都是由计算机网络自动完成的，而且以电或光的速度传播，所以速度非常快。传递路径上转发机构（通常是路由器）的多少决定了一条消息到达对方计算机所消耗的时间（术语称延时），因此位于深圳的用户给美国的朋友发送消息有可能比给武汉的朋友发送消息还快。实时输入/输出的软件对计算机网络的延时要求较高，尤其是实时强交互性软件，对计算机网络延时要求更苛刻；而批量输入/输出的软件对计算机网络的延时要求并不高，比如在线看电影，刚开始有点延时，后面就一直流畅地播放了。

一个云计算中心的延时半径通常为 100 毫秒，即一个数据包从云中心出发，50 毫秒所能到达的范围（返回也要 50 毫秒），这个数字与地理位置没有直接关系，而与网络路径上的转发机构和数目有关。比如深圳的超算中心 50 毫秒延时半径可能包括了美国的洛杉矶，但没有包括广东省内的梅州市，因为深圳与梅州市之间要经过太多性能低下的转发设备，而达到美国洛杉矶只经过少数几台高速路由器，如图 2-7 所示。

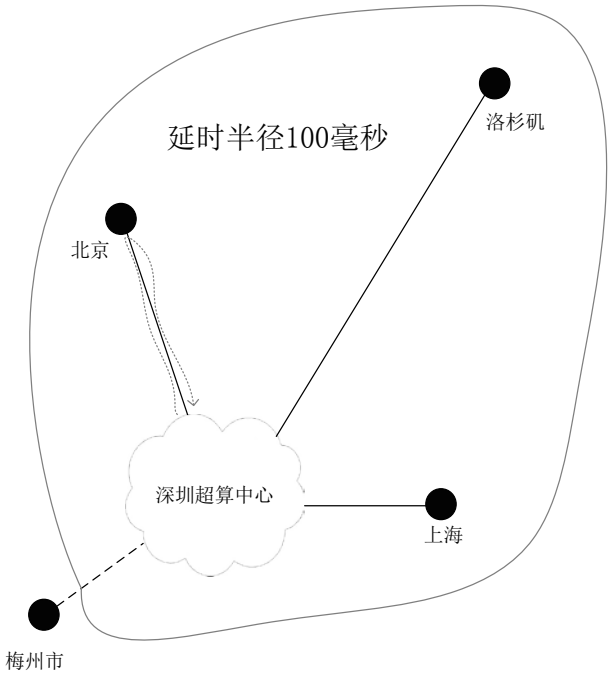


图 2-7　延时半径 100 毫秒

计算机网络的另一个指标是带宽，定义为每秒钟能传递的数据量。带宽越大，则每秒钟传递的数据量就越大。如果把计算机网络比喻为布满收费站的高速公路（车子从一个收费站行驶到另一个收费站的时间可以忽略不计，但是在收费站交钱时要消耗时间），那么带宽就与车道数直接相关，延时就是从出发点到目的地经过收费站交钱时所消耗的时间之和。

计算机网络的第三个指标是丢包率（或称为掉包率），是指在一定时间内被丢掉的数据包数目占总发送数据包数目的百分比。公式如下：

$$\text{丢包率}=\frac{\text{本机发出的数据包总数}-\text{对方收到的数据包总数}}{\text{本机发出的数据包总数}}\times 100\%$$

本机发送的一个数据包如果在规定的时间内没有得到对方的确认，就认为此数据包被丢失，于是重发此数据包。比如对方一共收到 8 个数据包，而本机一共发送了 10 个数据包（因为重发了 2 个数据包），所以丢包率为 20%，

即$\frac{10-8}{10}\times100\%$。丢包率越小，说明网络越稳定，当丢包率超过 10%时就很严重了，这时需要检查网络。

接下来我们再来谈谈叠加网络技术。为了方便大家去网上搜索，在这里给出这个术语的英文名称：Overlay Networks，在谷歌上搜索英文的技术资料，往往能获得很多你想要的结果。叠加网络，顾名思义就是在一张网络平面上叠加出更多层的网络平面，手法无非就是“包中之包”——把叠加协议和信息数据打包，作为底层网络平面传递的应用层数据。实现叠加的最新技术有 VXLAN、NVGRE 和 STT，这些技术主要用来解决在大规模、多机房、跨地区的云计算中心部署多租户环境问题。叠加网络技术非常类似于在邮政系统上建立一个情报网络，利用现有的邮局来收发情报。为了防止未经授权的人获取情报信息，需要先做加密处理，然后再放入信封，当然对方事先要知道解密方法。

叠加网络与虚拟局域网有着本质的不同，虚拟局域网通过分割一张大的局域网来减少广播风暴，本质是“分割”网络；而叠加网络是把“局域网”延伸到底层网络平面的任何地点，本质是“连通”。这里的“局域网”是虚拟的概念，对用户是透明的。VPN 就是一个典型的在广域网中通过叠加网络技术构建地理位置跨度很大的局域网的例子，如果不采用叠加网络技术，那么一个公司很难或者根本不可能建立一个跨城市的局域网。为了进一步说明叠加网络的概念，请看下面的多租户环境的情景案例。

中国微算科技有限公司运营一个大型公有云，包含北京、上海和西安三个计算中心，共拥有 80 万台服务器，构造出近 5000 万台虚拟机，主营业务是对外出租虚拟机（IaaS 云服务）。中国慧献是一家全国性的集团公司，机构分布在中国各地，为了节约成本和快速部署应用，该公司向中国微算科技有限公司长期租赁 1 万台虚拟机，因而成为中国微算科技有限公司的最大租户。这 1 万台虚拟机由中国慧献集团内部的技术工程师规划成近 300 个局域网（VXLAN），每个局域网内的虚拟机个数相对固定，但是虚拟机运行的地点与

使用虚拟机的员工紧密相关。比如中国慧献员工郭淑敏在北京出差十天，她的虚拟机就在北京的云中心运行，等她返回上海后，虚拟机也将“漂移”到上海的云中心运行。这一切，对郭淑敏来说都是透明的，她只是感觉首次在北京登录云端时稍微慢一点，几分钟之后就很流畅了，首次在上海登录时也是如此。不管在哪里，郭淑敏都是登录相同的 IP 地址，输入相同的账户和密码，然后看到自己熟悉的计算机桌面。是的，昨天编辑的 PPT 继续打开在桌面上，光标停在“公司今年的财务”处，郭淑敏继续输入“状况好于去年……”郭淑敏的虚拟机归口于财务部局域网，财务部局域网横跨北京、上海和西安云中心，这是一个典型的叠加网络案例。

最后总结一下本节的知识点：

（1）计算机网络解决位于不同地区的两台计算机之间如何通信的问题。

（2）需要与外界通信的计算机必须拥有一个唯一的 IP 地址。

（3）计算机网络的三个重要指标是带宽、延时和丢包率。

（4）延时由网络路径上的转发机构的速度和数量决定，与通信双方地理位置的远近无关。

（5）叠加网络技术解决了在广域网上灵活构建虚拟局域网的问题。

我们要重点关注带宽和延时，因为它们是部署云计算时不可忽视的两个重要因素。

2.4 IT 系统组成

当今，任何一家企业，无论规模大小，都要采用计算机来处理日常事务，如写文档、发邮件、管理库存、管理客户等。为此，企业需要建设计算机网

络，购买计算机设备，安装各种平台软件和应用软件。随着公司的发展壮大，企业中的计算机网络变得越来越复杂，计算机设备越来越多，安装的软件也五花八门，到后来各种问题就出现了：病毒肆虐、数据丢失、客户流失、源代码被盗、网速下降、数据孤岛难以共享、运维复杂等。为了搞清楚企业中复杂的 IT 系统结构，我们假设一家投资 8000 万元的公司诞生了，该公司购买了一栋办公楼，现在需要计算机工程师们把 IT 系统搭建起来。工程师们制订了如下工作计划：

（1）机房基础建设，包括机房选址、装修、供电、温/湿度控制、监控、门禁等。

（2）组建计算机网络，包括大楼综合布线、机柜安装、网络设备购买安装和调试。

（3）安装存储磁盘柜。

（4）购买和配置服务器。注意，还可能是虚拟服务器。

（5）安装操作系统。

（6）安装数据库。

（7）安装各种中间件和运行库。

（8）安装各种应用软件。

（9）导入公司的初始化业务数据。

至此，公司的整体 IT 系统搭建完毕，员工可以入驻办公了。根据上面的工作计划，我们可以很容易地总结出企业 IT 系统的逻辑层次结构，如图 2-8 所示。

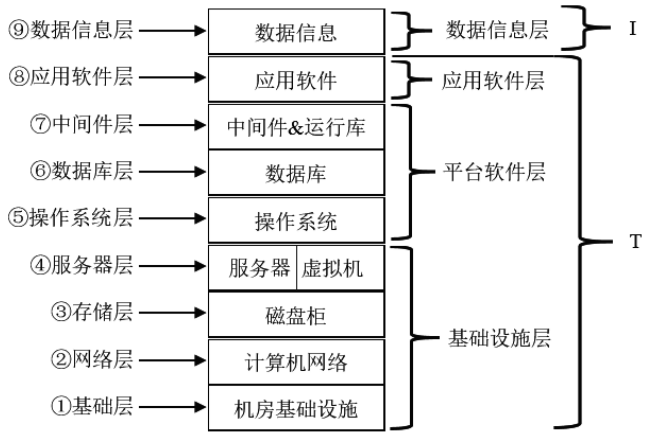


图 2-8　IT 系统的逻辑层次结构

一个典型的 IT 系统从逻辑上分为 9 层，施工时也是严格按照从第 1 层到第 9 层的顺序进行的，这就是所谓的“竖井”式施工。其中，第 1 ~ 4 层可归并为基础设施层，第 5 ~ 7 层可归并为平台软件层。九层归并之后形成四层结构，分别是基础设施层、平台软件层、应用软件层和数据信息层，IT 系统的四层结构是最为普遍并被广泛接受的划分方法。在后续章节中，我们将采用四层结构展开讨论。

基础设施层、平台软件层、应用软件层可以进一步归并到 T（Technology 的首字母，表示技术），而数据信息层就是 I（Information 的首字母，表示信息），这就是 IT 的含义——信息技术。对于一家企业而言，随着时间的推移，积累的数据信息会越来越多，数据信息是企业的宝贵资产，甚至是关乎企业生死存亡的重要财富。“如果数据丢失，80%的企业要倒闭。”此话并非是危言耸听。信息是目的，技术只是手段，如果一家企业没有业务数据需要处理，那么花大量资金组建基础设施层、平台软件层、应用软件层又有什么意义呢?

记住：IT 就是信息（Information）与技术（Technology），其中 I 是目的，T 是手段，T 是用来加工处理 I 的。T 广义上还包括企业中的计算机技术人员。

这里要重点介绍平台软件层的作用。很多计算机专业人士对平台软件难以理解，平台软件存在的唯一理由就是让应用软件能在计算机上运行。换句

话说，平台软件就是应用软件运行时所依赖的环境。比如，要想使用 QQ 这个应用软件，就必须先安装操作系统（如 Windows 10），QQ 需要的运行库在安装操作系统时会自动安装，然后才可以安装并运行 QQ 和朋友聊天。应用软件与平台软件的关系如图 2-9 所示。

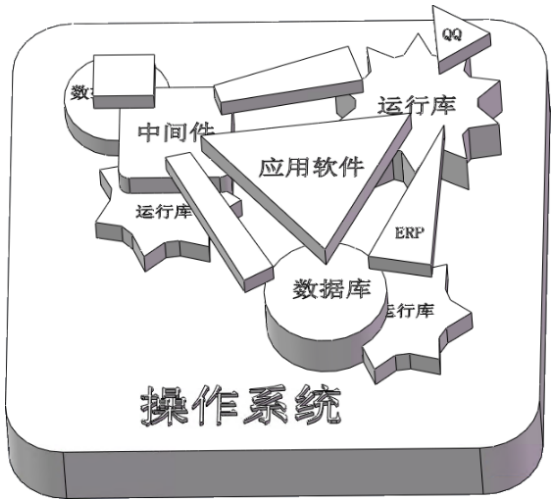


图 2-9　应用软件与平台软件的关系

从图 2-9 可知，在操作系统平台上再搭建运行库、中间件和数据库三个“台”，最后在最上层放置应用软件。不过，中间件可能还需要运行库和数据库的支撑，数据库可能还需要运行库的支撑。并不是每个应用软件都要同时压在中间件、运行库和数据库三个“台”上，有的应用软件只需要运行库（如 QQ），有的应用软件只需要中间件，有的应用软件同时需要运行库和数据库，但是不需要运行库的应用软件很少，静态编译的软件运行时不需要运行库。

运行库有点像电工人员的工具袋，里面有螺丝刀、电笔、老虎钳、剥线钳等，应用软件在运行时需要使用各种小工具（术语叫系统库函数调用），操作系统提供了绝大多数常用的小工具，并分门别类地保存在硬盘的文件中。Windows 操作系统中以.dll 为扩展名的文件，通常保存在 C:\Windows\System32 下（比如文件 GDI32.dll 就是 QQ 软件运行时要用到的工具箱之一，如果把此文件删除，那么 QQ 就无法运行）；Linux 操作系统中一般以.so 作为扩展名，

保存在/lib 目录下。不同操作系统提供的“工具”和使用方法也不同，所以能在 Windows 操作系统上运行的应用软件不能在 Linux 操作系统上运行，也不能在苹果公司的 Macintosh 操作系统上运行，反之亦然。为此，应用软件开发商会针对不同的操作系统发行不同的软件版本。

“中间件”是一个技术含义很强的概念，在家庭计算机和个人计算机上很少用到它，在企业中使用很普遍。中间件也就是“中间软件”的意思，为一类软件的统称。“中间”包含两方面含义：一是指处于操作系统和应用软件之间；二是指介于应用软件与应用软件之间，目的是为了隐藏差异，以便共享资源和通信。中间件有点类似于电源插座面板，不管插座里面是什么构造，面板上的插接孔都是一样的，这样插座面板一方面隐藏了插座内部结构，另一方面能接插所有的电源插头。中间件的引入目的就是要隐藏通信对端的内部结构，而呈现统一的调用界面。

掌握企业 IT 系统的四层逻辑结构很重要，这对于理解后续章节中的云计算概念非常有帮助。

2.5 云计算概念

在本章 2.1 节中，我们谈到，软件就是程序员写的需要 CPU 来执行以便完成某项任务的步骤，这些步骤包括输入/输出步骤和计算步骤。而 CPU 在执行输入/输出步骤时需要使用输入/输出设备，在执行计算步骤时需要使用计算设备。对于普通的计算机来说，计算设备指 CPU、内存和硬盘，输入/输出设备指键盘、鼠标、显示器、话筒和音箱等。如果计算机还要与其他设备通信，那么计算设备还应包括网络。对于传统的个人计算机而言，计算设备和输入/输出设备通过主板连接在一起，也就是说，有了主板这个纽带，计算设备和输入/输出设备就可以协同工作了。其特征之一是计算资源就在本地；特征之二是计算资源不易扩展或收缩；特征之三是其他人无法共享你的计算资源；特征之四是你既是计算资源的所有者，又是计算资源的使用者。

我们再次重申一下：计算设备也称为计算资源，计算资源包括 CPU、内存、硬盘和网络。而在机房中，磁盘只是存储大类中的一种，存储器还包括磁带库、阵列、SAN、NAS 等，这些统称为存储资源。另外，CPU、内存只是服务器的部件，我们统一用服务器资源来代替 CPU 和内存资源的说法。广义的计算资源还包括应用软件和人力服务，如果不特别声明，那么后续章节中提到的计算资源就是指服务器、存储器、网络、应用软件和人力服务。

不同于传统的计算机，云计算引入了一种全新的方便人们使用计算资源的模式，类似人们洗衣服由使用洗衣机转变为去洗衣店那样，即：

云计算是一种全新的能让人们方便、快捷地自助使用远程计算资源的模式。

计算资源所在地称为云端（也称为云基础设施），输入/输出设备称为云终端。云终端就在人们触手可及的地方，而云端位于“远方”（与地理位置远近无关，需要通过网络才能到达），两者通过计算机网络连接在一起。云终端与云端之间是标准的 C/S 模式，即客户端/服务器模式——客户端通过网络向云端发送请求消息，云端计算处理后返回结果。云计算的可视化模型如图 2-10 所示。

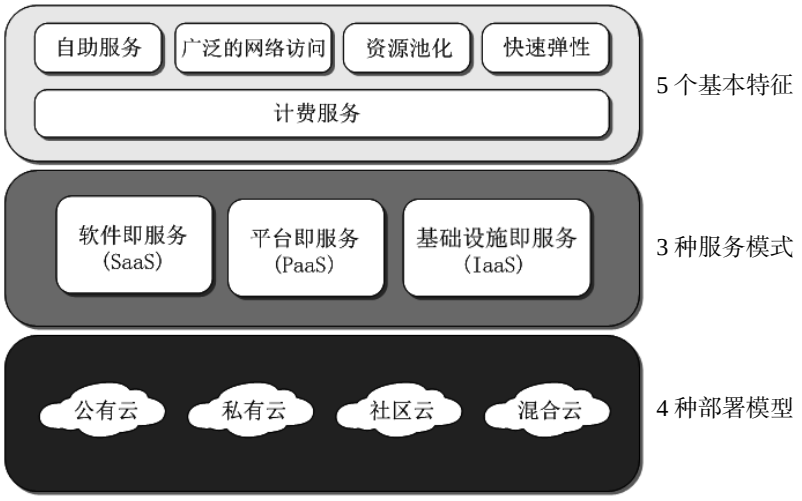


图 2-10　云计算的可视化模型

云计算具有 5 个基本特征、4 种部署模型和 3 种服务模式。

云计算的 5 个基本特征如下：

（1）自助服务。消费者不需要或很少需要云服务提供商的协助，就可以单方面按需获取并使用云端的计算资源。

（2）广泛的网络访问。消费者可以随时随地使用任何云终端设备接入网络并使用云端的计算资源。常见的云终端设备包括手机、平板电脑、笔记本电脑、PDA 掌上电脑和台式计算机等。

（3）资源池化。云端计算资源需要被池化，以便通过多租户形式共享给多个消费者，也只有池化才能根据消费者的需求动态分配或再分配各种物理的和虚拟的资源。消费者通常不知道自己正在使用的计算资源的确切位置，但是在自助申请时允许指定大概的区域范围（比如在哪个国家、哪个省或者哪个数据中心）。

（4）快速弹性。消费者能方便、快捷地按需获取和释放计算资源，也就是说，需要时能快速获取资源从而扩展计算能力，不需要时能迅速释放资源以便降低计算能力，从而减少资源的使用费用。对于消费者来说，云端的计算资源是无限的，可以随时申请并获取任意数量的计算资源。但是我们一定要消除一个误解，那就是一个实际的云计算系统不一定是投资巨大的工程，也不一定要购买成千上万台计算机，也不一定具备超大规模的运算能力。云端建设方案一般采用可伸缩性策略，刚开始时采用几台计算机，以后根据用户数量规模来弹性增减机器数量。

（5）计费服务。消费者使用云端计算资源是要付费的，付费的计量方法有很多，比如根据某类资源（如存储、CPU、内存、网络带宽等）的使用量和时间长短计费，也可以按照每使用一次多少钱来计费。但不管如何计费，对消费者来说，价码要清楚，计量方法要明确，而云服务提供商需要监视和控制资源的使用情况，并及时输出各种资源的使用报表，做到供/需双方费用结算清楚明白、准确无误。

云计算的 4 种部署模型如下：

（1）私有云。云端资源只给一个单位组织内的用户使用，这是私有云的核心特征。而云端的所有权、日常管理和操作的主体到底属于谁并没有严格的规定，可能是本单位，也可能是第三方机构，还可能是二者的联合。云端可能位于本单位内部，也可能托管在其他地方。

（2）社区云。云端资源专门给固定的几个单位内的用户使用，而这些单位对云端具有相同的诉求（如安全要求、云端使命、规章制度、合规性要求等）。云端的所有权、日常管理和操作的主体可能是本社区内的一个或多个单位，也可能是社区外的第三方机构，还可能是二者的联合。云端可能部署在本地，也可能部署于他处。

（3）公有云。云端资源开放给社会公众使用。云端的所有权、日常管理和操作的主体可以是一个商业组织、学术机构、政府部门或者它们其中的几个联合。云端可能部署在本地，也可能部署于其他地方，比如中山市民公有云的云端可能就建在中山，也可能建在深圳。

（4）混合云。混合云由两个或两个以上不同类型的云（私有云、社区云、公有云）组成，它们各自独立，但用标准的或专有的技术将它们组合起来，而这些技术能实现云之间的数据和应用程序的平滑流转。由多个相同类型的云组合在一起属于多云的范畴，比如两个私有云组合在一起，混合云属于多云的一种。由私有云和公有云构成的混合云是目前最流行的——当私有云资源短暂性需求过大（称为云爆发）时，自动租赁公有云资源来平抑私有云资源的需求峰值。例如，网店在节假日期间点击量巨大，这时就会临时使用公有云资源来应急。

云计算的 3 种服务模式如下：

（1）软件即服务（Software as a Service，SaaS）。云服务提供商把 IT 系统中的应用软件层作为服务出租出去，消费者不用自己安装应用软件，直接使

用即可，这进一步降低了云服务消费者的技术门槛。更详细的介绍参见后续章节。

（2）平台即服务（Platform as a Service，PaaS）。云服务提供商把 IT 系统中的平台软件层作为服务出租出去，消费者自己开发或者安装程序，并运行程序。更详细的介绍参见后续章节。

（3）基础设施即服务（Infrastructure as a Service，IaaS）。云服务提供商把 IT 系统的基础设施层作为服务出租出去，由消费者自己安装操作系统、中间件、数据库和应用程序。更详细的介绍参见后续章节。

云计算的精髓就是把有形的产品（网络设备、服务器、存储设备、各种软件等）转化为服务产品，并通过网络让人们远距离在线使用，使产品的所有权和使用权分离。正如洗衣店老板把洗衣机这种有形产品转化为洗衣服务一样，消费者直接投币自助洗衣，这样一方面提高了洗衣机的使用率，另一方面降低了消费者购买洗衣机的支出。洗衣店不是一种新的洗衣服技术，而是一种新的洗衣服模式，可能洗衣店老板还会进一步细分市场，推出干洗烘干一条龙服务等。同样，云计算也不是一种新的计算技术，而是一种新的计算模式。计算设备一旦转化为服务，使用率就会得到显著提高，但设备的寿命并没有缩短反而会更长，因为电子产品不同于机械产品会产生磨损，经常不开机的电子设备会比常年开机满负荷运转的电子设备更容易出故障。另外，作为计算资源的软件可以无限复制运行，这一点与洗衣店的洗衣机等物理设备有本质的不同，一台洗衣机被他人使用，就不能同时再被另外的人使用；而软件则没有这个限制，同样一个软件，可以同时给任意多的人使用。还有，计算资源使用率的高低与成本无关，对于一台计算机而言，直接的成本是用电费用，计算机空转与满负荷运转耗费的电力几乎相当；而洗衣店的用户越多，开启的洗衣机就越多，耗费的电力也就越多。目前，世界范围内传统计算设备远远超过云端的计算设备，这些传统的计算设备由于没有共享，因此很多计算产能被白白浪费掉了。加入云端的计算设备在得到充分使用的情况

下，云计算才算是“绿色”计算，但是目前几大云计算中心，计算设备的利用率却非常低，机器众多、耗电巨大，这根本就是与“绿色”计算背道而驰。

通过云计算，提供商把计算资源转化为服务产品并销售给用户，服务产品有别于其他的有形产品（如空调、桌子、啤酒、书籍等），所以有人提出了云计算的第 4 种服务模式——DaaS（数据即服务，就是出租 IT 系统的最顶层）。这等同于说空调即服务、桌子即服务，这是不恰当的，因为数据是一种有形产品（以纸张、U 盘、磁盘作为载体，正如书籍是知识的载体），而且在目前的技术条件下，几乎不可能把数据转化为服务。服务的一个主要属性是所有权和使用权分离，使用一次服务能预先摊算费用。而用户一旦使用了一次数据，其实他就拥有了该数据，此后他可以无限制使用，而且还可能传播和转卖。

前面多次提到租户和用户，那么这两个概念到底有什么区别呢？以一个单位组织（如企业、部门或团体等）的名义去租赁云计算服务时，云服务提供商称呼该单位组织为一个租户，而一个租户包含若干个用户（单位内的员工），这些用户当中有的是该租户的管理员，有的是操作员等。比如 XXX 企业向 SaaS 云服务提供商租赁了 ERP 系统，云服务提供商认为 XXX 企业是一个租户，然后给它分配管理员账号和密码。XXX 企业指定员工张三为租户管理员，张三登录云平台自助网站创建更多的普通账号，然后把这些账号分配给公司内部的相关业务人员（如会计人员、人事经理、仓库管理员等）。一个租户允许只有一个用户，也允许有多个用户。云服务提供商只与租户（法人代表）之间存在租赁合同关系，并与租户进行费用结算，云服务提供商与租户关系，如图 2-11 所示。

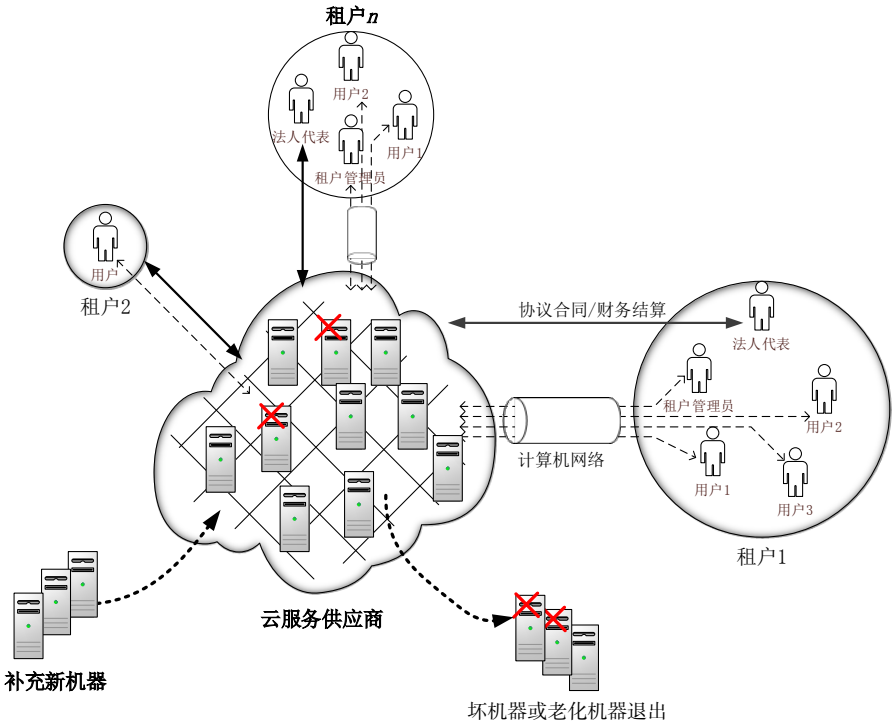


图 2-11　云服务提供商与租户关系

2.6　3 种服务模式

在本章 2.4 节中讲到，IT 系统的逻辑组成分为四层，自下至上依次是基础设施层、平台软件层、应用软件层和数据信息层。云计算是一种新的计算资源使用模式，云端本身还是 IT 系统，所以逻辑上同样可以划分为这四层。底三层可以再划分出很多“小块”并出租出去，这有点像立体停车房，按车位大小和停车时间长短收取停车费。因此，云服务提供商出租计算资源有 3 种模式，满足云服务消费者的不同需求，分别是 IaaS、PaaS、SaaS，如图 2-12 所示。

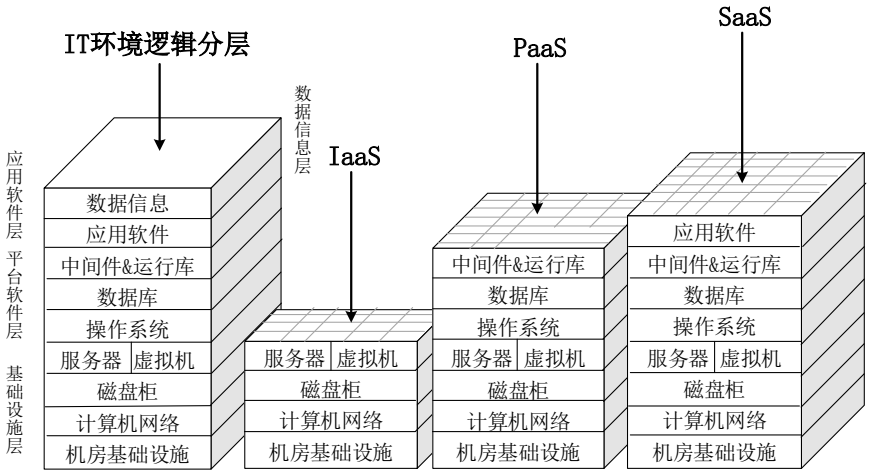


图 2-12　云计算的 3 种服务模式

需要注意的是，云服务提供商只负责出租层及以下各层的部署、运维和管理，而租户自己负责更上层次的部署和管理，两者负责的“逻辑层”加起来刚好就是一个完整的四层 IT 系统（见图 2-12 最左侧）。比如有一家云服务提供商对外出租 IaaS 云计算业务，云服务提供商负责机房基础设施、计算机网络、磁盘柜和服务器/虚拟机的建设和管理，而云服务消费者自己完成操作系统、数据库、中间件&运行库和应用软件的安装和维护。另外，还要管理数据信息（如初始化、数据备份、恢复等）。再比如，另一家云服务提供商出租 PaaS 业务，那么云服务提供商负责的层数就更多了，云服务消费者只需安装自己需要的应用软件并进行数据初始化即可。总之，云服务提供商和消费者各自管理的层数加起来就是标准的 IT 系统的逻辑层次结构。

下面对这 3 种服务模式分别做进一步的介绍。

2.6.1　IaaS

IaaS 是 Infrastructure as a Service 的首字母缩写，意思是基础设施即服务，即把 IT 系统的基础设施层作为服务出租出去。由云服务提供商把 IT 系统的基础设施建设好，并对计算设备进行池化，然后直接对外出租硬件服务器、

虚拟主机、存储或网络设施（负载均衡器、防火墙、公网 IP 地址及诸如 DNS 等基础服务）等。云服务提供商负责管理机房基础设施、计算机网络、磁盘柜、服务器和虚拟机，租户自己安装和管理操作系统、数据库、中间件&运行库、应用软件和数据信息，所以 IaaS 云服务的消费者一般是掌握一定技术的系统管理员，如图 2-13 所示。

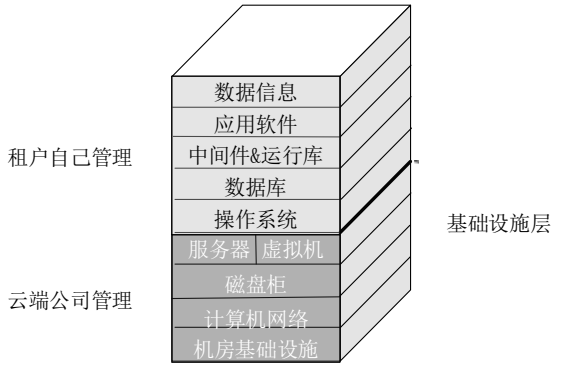


图 2-13　IaaS 云服务

IaaS 云服务提供商计算租赁费用的因素包括 CPU、内存和存储的数量，一定时间内消耗的网络带宽，公网 IP 地址数量及一些其他需要的增值服务（如监控、自动伸缩等）。

出租的物理服务器和虚拟机统称为主机，云服务提供商如何对外出租主机呢？或者是租户如何使用这些租来的主机呢？相对于租户来说，这些主机不在现场而在“远方”，租赁之后并不是把这些主机从云端搬到租户的办公室来使用。出租前后主机的物理位置并没有改变，租户仍然是通过网络使用这些云端主机。租户登录云服务提供商的网站，填写并提交主机配置（如需要多少个 CPU、多少内存、多少网络带宽等）表后付款，然后云服务提供商向租户颁发账号和密码，最后租户以此账号和密码登录云端的自助网站。在这里，租户可以管理自己的主机：启动和关闭机器、安装操作系统、安装和配置数据库、安装应用软件等。其实只有启动机器和安装操作系统必须要在自助网站上完成，其他操作可以直接登录到已经安装了操作系统并配置好网卡

的主机中完成。对于租来的主机，租户只关心计算资源（CPU、内存、硬盘）的容量是否与租赁合同上标注的一致，就像租赁同一层楼上的一间房间一样，租户只关心面积是否足够，而不关心房间的墙壁是钢筋水泥结构还是砖块石灰结构。但是对于云服务提供商来说，出租硬件服务器和虚拟机，内部的技术处理是不一样的，其中硬件服务器必须要集成远程管理卡并要池化到资源池中。

远程管理卡是插接在服务器主板上或者直接集成在主板上的一个嵌入式系统，需要接网线并配置 IP 地址。只要服务器的电源插头插到插座上，不管有没有按下服务器的电源开关，这个远程管理卡都会启动，其他人就可以通过网络登录到远程管理卡（需要账号和密码），成功登录后就可以进行启动和关闭服务器、安装操作系统等操作。只不过云服务提供商把远程管理卡的功能集成到了租户自助网站中，从而实现了物理机和虚拟机的统一管理。租户到底是租赁硬件服务器还是虚拟机呢？这个问题留到第 3 章来讨论。

IaaS 云端的基本架构模型如图 2-14 所示。

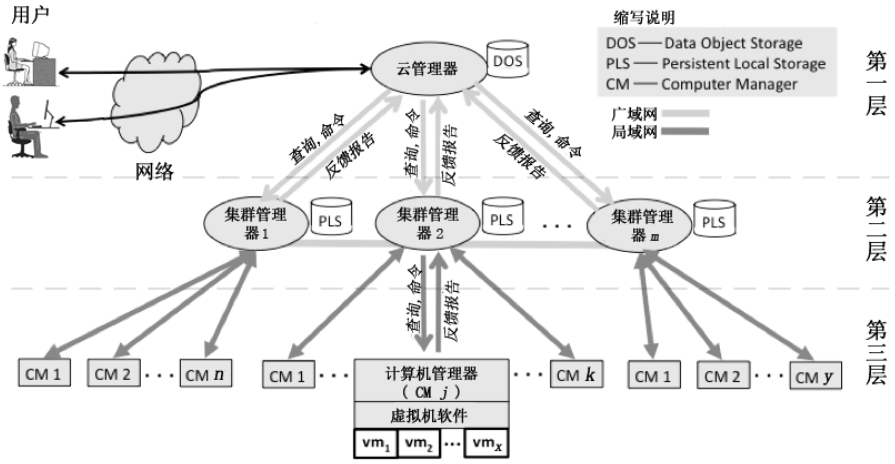


图 2-14　IaaS 云端的基本架构模型

IaaS 云端的基本架构模型逻辑上分为三层：第一层管理全局，第二层管理计算机集群（一个集群内的机器地理位置上可能相距很远），第三层负责运行虚拟机。第一层的云管理器与第二层的集群管理器之间一般通过高速网络

连接，当增加数据中心为云端扩容时，就能体现网速的重要性。而集群内的计算机之间倾向于采用本地局域网（如 10Gbit/s 以太网）或者超高速广域网，如果采用局域网，则灾难容错差；如果跨广域网，则网络带宽会成为瓶颈。

图 2-14 中，每一层具体的任务介绍如下。

第一层（云管理器）：云管理器是云端对外的总入口，在这里验证用户身份，管理用户权限，向合法用户发放票据（然后用户持此票据使用计算资源）、分配资源并管理用户租赁的资源。

第二层（集群管理器）：每一个集群负责管理本集群内部的高速互联在一起的计算机，一个集群内的计算机可能有成百上千台。集群管理器接收上层的资源查询请求，然后向下层的计算机管理器发送查询请求，最后汇总并判断是部分满足还是全部满足上层请求的资源，再反馈给上层。如果接下来收到上层分配资源的命令，那么集群管理器指导下层的计算机管理器进行资源分配并配置虚拟网络，以便能让用户后续访问。另外，本层 PLS 中存储了本集群内的全部虚拟机镜像文件，这样一台虚拟机就能在集群内任意一台计算机上运行，并轻松实现虚拟机热迁移。

第三层（计算机管理器）：每台计算机上都有一个计算机管理器，它一方面与上层的集群管理器打交道，另一方面与本机上的虚拟机软件打交道。它把本机的状态（如正在运行的虚拟机数、可用的资源数等）反馈给上层，当收到上层的命令时，计算机管理器就指导本机的虚拟机软件执行相应命令。这些命令包括启动、关闭、重启、挂起、迁移和重配置虚拟机，以及设置虚拟网络等。

租赁 IaaS 云服务，对租户而言，最大优点是其灵活性，由租户自己决定安装什么操作系统、需不需要数据库且安装什么数据库、安装什么应用软件、安装多少应用软件、要不要中间件、安装什么中间件等，相当于购买了一台计算机，要不要使用、何时使用，以及如何使用全由自己决定。一些搞研发的计算机技术人员倾向于租赁 IaaS 主机。但是对于租户来说，IaaS 云主机除

管理难度大外，还有一个明显的缺陷：计算资源浪费严重。因为操作系统、数据库和中间件本身要消耗大量的计算资源（CPU、内存和磁盘空间），但它们消耗的资源对租户来说是无用的，来看下面这个极端的案例。

假设张三租用了一台 IaaS 主机，配置为 CPU 1.0GHz，内存 1GB，硬盘空间 10GB，然后他安装了 Windows 7 操作系统、MySQL 数据库，最后再想安装和运行绘图应用软件几乎是不可能了，因为 Windows 7 操作系统和 MySQL 数据库就要把 CPU、内存和硬盘空间消耗殆尽了。没有硬盘空间，如何安装应用软件？没有空闲的内存，又如何能运行应用软件？张三本来打算租赁云端主机来搞图形设计，这样一来，目的就没达到，至少租来的主机的配置要翻倍才行。

下面是一些 IaaS 云服务的实际应用。

（1）备份和恢复服务。

（2）计算服务：提供弹性资源。

（3）内容分发网络（CDN）：把内容分发到靠近用户的地方，对于一些基于网页的应用系统，为了提高用户体验，往往在各个地方（人口稠密的地方）设立分支服务器，当用户浏览网页时，被重定向到本地 Web 服务器，所以数据必须实时分发并保持一致。

（4）服务管理：管理云端基础设施平台的各种服务。

（5）存储服务：提供用于备份、归档和文件存储的大规模可伸缩存储。

2.6.2 PaaS

PaaS 是 Platform as a Service 的首字母缩写，意为平台即服务，即把 IT 系统的平台软件层作为服务出租，如图 2-15 所示。

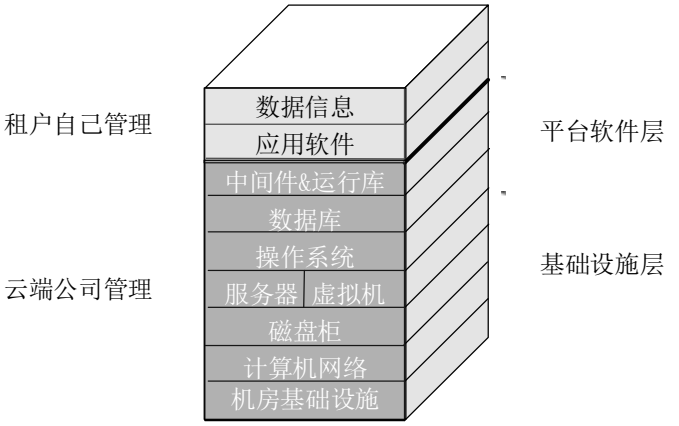


图 2-15　PaaS 云服务

相比于 IaaS 云服务提供商，PaaS 云服务提供商要做的事情增加了，它们需要准备机房、布好网络、购买设备、安装操作系统、数据库和中间件，即把基础设施层和平台软件层都搭建好，然后在平台软件层上划分“小块”（习惯称之为容器）并对外出租。PaaS 云服务提供商也可以从其他 IaaS 云服务提供商那里租赁计算资源，然后自己部署平台软件层。另外，为了让消费者能直接在云端开发调试程序，PaaS 云服务提供商还得安装各种开发调试工具。相反，租户要做的事情相比 IaaS 要少很多，租户只要开发和调试软件或者安装、配置和使用应用软件即可。PaaS 云服务的消费者主要包括以下人员。

（1）程序开发人员：写代码、编译、调试、运行、部署、代码版本控制等，一揽子解决。

（2）程序测试人员。

（3）软件部署人员：把软件部署到 PaaS 云端，便于管理不同版本之间的冲突。

（4）应用软件管理员：便于配置、调优和监视程序运行性能。

（5）应用程序最终用户：这时 PaaS 云服务相当于 SaaS 云服务。

PaaS 云服务的费用计算一般根据租户中的用户数量、用户类型（如开发

人员、最终用户等）、资源消耗量及租期等因素计算。图 2-16 所示描绘了 PaaS 云服务供/需双方的动态交互情况。

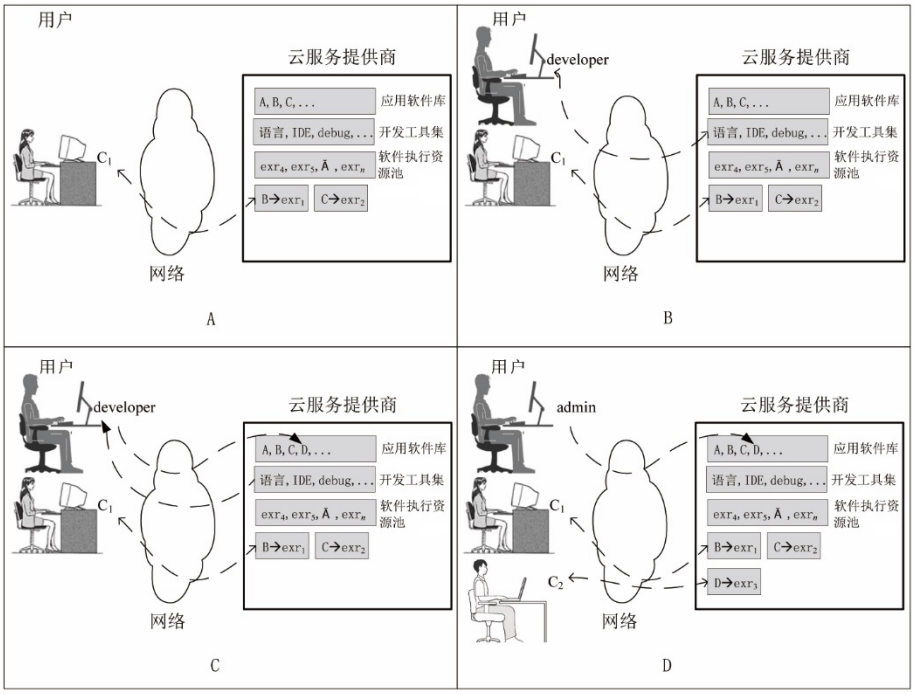


图 2-16　PaaS 云服务供/需双方动态交互情况

云服务提供商拥有一个应用软件库（图 2-16 中“A,B,C, …”代表库中的应用软件）、开发工具集（如编程语言、IDE、调试工具等）和软件执行资源池（图 2-16 中“exr_1, exr_2, …”代表资源），消费者通过网络远程租赁软件执行资源并运行相关的应用软件，或者使用各种开发工具。软件执行资源有“可用”和“占用”两个状态，同一个资源不能同时运行多个程序。软件执行资源可能是物理机、虚拟机、容器或者一个正在运行的服务程序（响应消费者的请求，启动虚拟机或容器，甚至是租赁其他云端的计算资源）。图 2-16 A 中，用户 C_1 申请了两个资源：exr_1 和 exr_2，并运行 B、C 两个程序；图 2-16 B 中，新来了一个程序开发人员，他正在使用开发工具开发程序；图 2-16 C 中，程序开发人员开发完成并部署了一个新程序 D；图 2-16 D 中，一个管理员发布

了程序 D，然后新来的用户申请资源 exr_3 并执行程序 D。PaaS 云服务供/需交互还有很多情景，这里不再一一列出。

比如我们要安装和使用 OpenERP 软件，这个应用软件要用到 PostgreSQL 数据库和 Python 语言，那么只需要租赁一个 PaaS 型容器并在里面安装 OpenERP 即可，但这个容器必须支持 PostgreSQL 数据库和 Python 语言，让租户无须再去安装和配置它们。

同样，我们可以租赁一个支持 PHP 语言和 MySQL 数据库的 PaaS 容器，然后采用 WordPress 开源建站工具，只需几步，就可以搭建一个个人博客网站。

应用软件数不胜数，支撑它们的编程语言、数据库、中间件和运行库可能都不一样。PaaS 云服务提供商不可能安装全部的语言、数据库、中间件和运行库来支持所有的应用软件，因此目前普遍的做法是安装主流的编程语言、数据库、中间件和运行库，使得出租的 PaaS 容器支持有限的、使用量排名靠前的应用软件，以及支持最流行的编程语言，并在网站上发布公告。当然，云服务提供商也鼓励租户直接开发支持 PaaS 的应用软件，每家云服务提供商都想尽可能地黏住更多的用户，这无可厚非。

不知道你有没有遭遇过这样的痛苦：想要安装一个软件，结果报“没有找到 XXX 中间件”的错误，于是急忙找来相关资料并安装需要的中间件，但又报错：“此中间件需要 YYY 数据库的支持”，那么只好安装数据库，却再次报错：“此数据库需要 ZZZ 运行库”，于是又去安装相应的运行库，但报错又出现了……最后你愤怒了：我不用这个应用软件总可以了吧？但你的计算机中已经安装了一大堆用不上的软件，硬盘快满了，于是又不得不去一个一个地卸载，卸载时又不断跳出调查表，询问你卸载的原因：是软件不够好吗？是售后服务跟不上吗？影响计算机速度吗……安装过 Linux 操作系统的人估计都会有这样的痛苦经历。

PaaS 的优势就是解决应用软件依赖的运行环境（如中间件、数据库、运行库等），其所依赖的软件全部由云服务提供商安装，所以当租户安装应用软

件时，就不会再出现连续报错的情况。应用软件就像歌唱家，当其唱歌表演时需要一个舞台，这个舞台就是由基础设施层和平台软件层堆叠起来的。

前面讲过，平台软件层包括操作系统、数据库、中间件和运行库四部分，但并不是说在具体搭建平台软件层时一定要安装和配置这四部分软件，需要哪部分，以及安装什么种类的平台软件要根据应用软件来定。比如一家只针对 PHP 语言开发（应用软件用 PHP 编写）的 PaaS 云服务提供商，就没必要安装类似 Tomcat 的中间件了。根据平台软件层中安装的软件种类多少，PaaS 又分为两种类型。

（1）**半平台 PaaS**：平台软件层中只安装了操作系统，其他的留给租户自己解决。最为流行的半平台 PaaS 应用是开启操作系统的多用户模式，为每个租户创建一个系统账号，并对他们做权限控制和计算资源配额管制。半平台 PaaS 更关注租户的类型，如研发型、文秘型等，针对不同类型的租户做不同的权限和资源配置。Linux 操作系统的多用户模式和 Windows 操作系统的终端服务都属于半平台 PaaS，私有办公云多采用半平台 PaaS。

（2）**全平台 PaaS**：全平台 PaaS 安装了应用软件依赖的全部平台软件（操作系统、数据库、中间件、运行库）。不同于半平台 PaaS，全平台 PaaS 是针对应用软件来做资源配额和权限控制的，尽管最终还需要通过账号实现。公有云多采用全平台 PaaS。

相对于 IaaS 云服务，PaaS 云服务消费者的灵活性降低了，租户不能自己安装平台软件，只能在有限的范围内选择。但优点也很明显，租户从高深烦琐的 IT 技术中解放出来，专注于应用和自己的核心业务。

下面是一些 PaaS 云服务的实际应用。

（1）商业智能（BI）：用于创建仪表盘、报表系统、数据分析等应用程序的平台。

（2）数据库：提供关系型数据库或者非关系型数据库服务。

（3）开发和测试平台。

（4）软件集成平台。

（5）应用软件部署：提供应用软件部署的依赖环境。

2.6.3 SaaS

SaaS 是 Software as a Service 的首字母缩写，意为软件即服务。简言之，就是软件部署在云端，让用户通过网络来使用它，即云服务提供商把 IT 系统的应用软件层作为服务出租，而消费者可以使用任何云终端设备接入计算机网络，然后通过网页浏览器或者编程接口使用云端的软件。这进一步降低了租户的技术门槛，应用软件也无须自己安装了，而是直接使用软件，如图 2-17 所示。

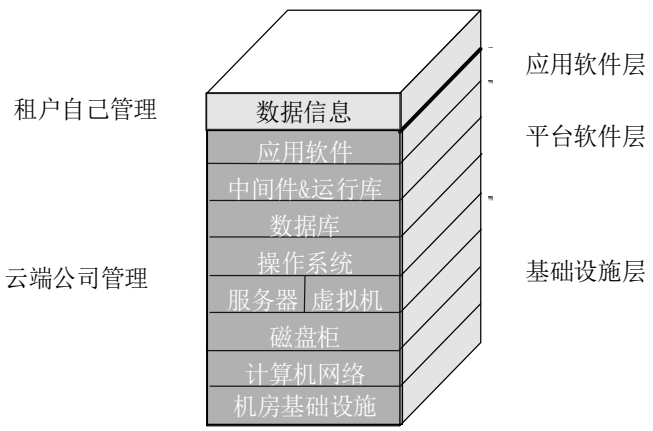


图 2-17　SaaS 云服务

这时 SaaS 云服务提供商有 3 种选择：

（1）租用别人的 IaaS 云服务，自己再搭建和管理平台软件层和应用软件层。

（2）租用别人的 PaaS 云服务，自己再部署和管理应用软件层。

（3）自己搭建和管理基础设施层、平台软件层和应用软件层。

总之，从云服务消费者的角度来看，SaaS 云服务提供商负责 IT 系统的底三层（基础设施层、平台软件层和应用软件层），也就是整个 T 层，最后直接把应用软件出租。图 2-18 所示是 SaaS 云服务供/需双方的动态交互示意图。

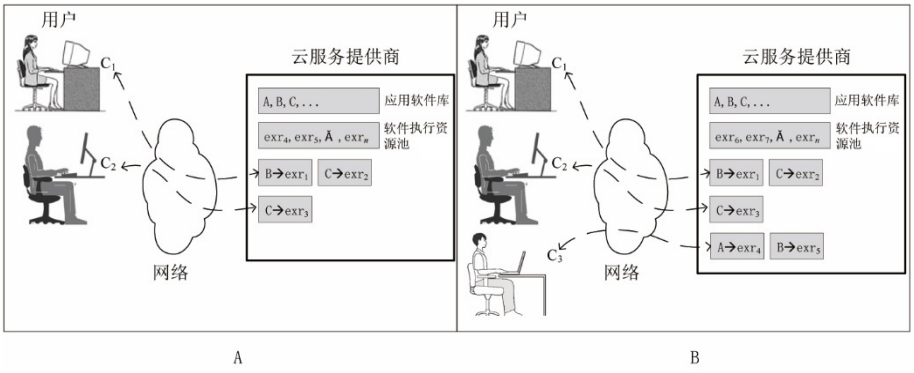


图 2-18　SaaS 云服务供/需双方动态交互示意图

云服务提供商拥有一个应用软件库（图 2-18 中“A,B,C,…”代表库中的应用软件）和应用软件执行资源池（图 2-18 中“exr_1, exr_2, …”代表资源），消费者通过网络远程租赁软件执行资源并运行相关的应用软件。在图 2-18 A 中，有两个用户正在使用云端的软件，其中用户 C_1 运行两个程序 B 和 C，云端为该用户分配了 exr_1 和 exr_2 两个软件执行资源，exr_1 软件执行资源用于执行 B 应用程序（图 2-18 中用“B→exr_1”表示），exr_2 软件执行资源用于执行 C 应用程序（图 2-18 中用“C→exr_2”表示）。而用户 C_2 正在执行一个程序 C（图 2-18 中用“C→exr_3”表示）。在图 2-18 B 中，一个新用户 C_3 进来了，云端从可用的软件执行资源池中为他分配了 exr_4 和 exr_5 两个执行资源，分别执行 A 和 B 应用程序，而软件执行资源池中可用的资源数减少了两个。

云服务提供商选择若干种使用面广且有利可图的应用软件，如 ERP（企业资源计划）、CRM（客户关系管理）、BI（商业智能）等，并精心安装和运维，让租户用得放心、安心。适合做 SaaS 的应用软件有以下几个特点。

（1）复杂。软件庞大、安装复杂、使用复杂、运维复杂，单独购买价格

昂贵，如 ERP、CRM 系统及可靠性工程软件等。

（2）主要面向企业用户。

（3）模块化结构。按功能划分成模块，租户需要什么功能就租赁什么模块，也便于按模块计费，如 ERP 系统划分为订单、采购、库存、生产、财物等模块。

（4）多租户。能适合多个企业中的多个用户同时操作，也就是说，使用同一个软件的租户之间互不干扰。租户一般指单位组织，一个租户包含多个用户。

（5）多币种、多语言、多时区支持。这一点对于公有云尤其明显，因为其消费者来自五湖四海。

（6）非强交互性软件。如果网络延时过大，那么强交互性软件作为 SaaS 对外出租就不太合适，会大大降低用户的体验度，除非改造成弱交互性软件或者批量输入/输出软件，如微软的 Office 365 和谷歌的在线办公等——通过浏览器运行远程 SaaS 办公软件，本质上就是 I/O 本地化，而浏览器与云端之间批量化传输（单击“保存”按钮或者浏览器定时保存）。

软件的云化就是对传统应用软件进行改造，使之满足（3）、（4）、（5）三个特点。

这里要着重介绍一下特点“（4）多租户”，即允许多个租户同时使用软件而互不影响，因此多租户的第一个要求就是软件支持多个用户登录，用户一般为非系统管理账户（如不是操作系统用户或者数据库用户），且保存在数据库的业务表中。在 2.1 节中已经讲过，软件就是由程序员写的让 CPU 来完成某项任务的步骤，包括输入/输出步骤和计算步骤，只有输入/输出步骤与多租户的特点相关，如张三的输入/输出就是张三的，绝对不允许“窜”到李四那边去，反之亦然。这里重点关注的是硬盘文件作为输入/输出设备的情景，因此多租户的第二个要求就是用户身份信息必须作为数据记录的检索字段之一，这样用户之间的数据才可以实现隔离。数据记录包括软件的配置信息和

业务数据，配置信息指租户选择的语言、设置的时区、指定的币种、定义的面板参数等；而业务数据就是日常业务处理产生的数据，一般保存在数据库中，而数据库保存在云端的存储中。在具体设计隔离方案时，需要综合考虑隔离效果和资源使用效率，如图 2-19 和图 2-20 所示，更详细的租户隔离内容参见后面的章节。

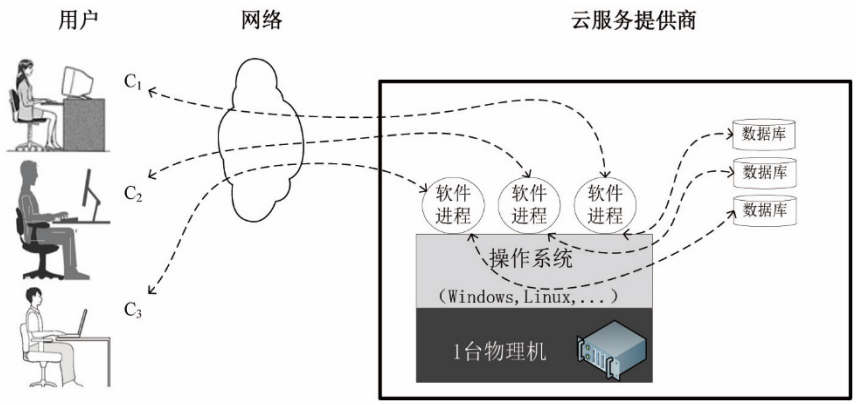


图 2-19　注重隔离效果

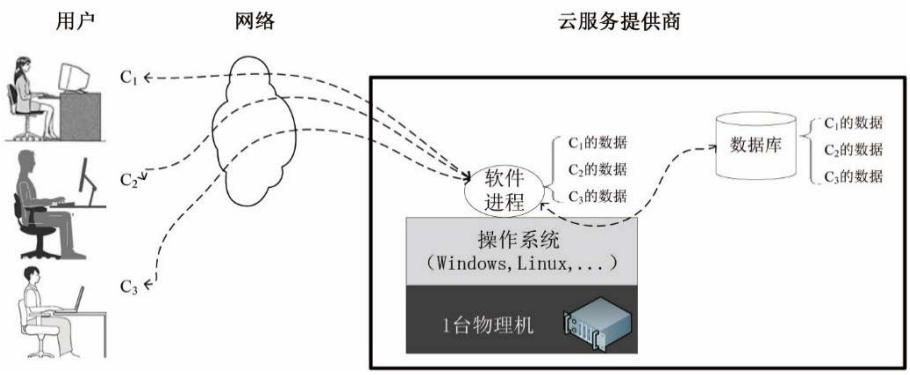


图 2-20　注重资源使用效率

下面这些类型的软件适合云化并以 SaaS 模式交付给用户。

（1）企事业单位的业务处理类软件：这类软件一般被单位组织用来处理提供商、员工、投资者和客户相关的业务，如开具发票、资金转账、库存管理及客户关系管理等。

（2）协同工作类软件：这类软件用于团队人员一起工作，团队成员可能都是单位组织内部的员工，也可能包含外部的人员。例如日历系统、邮件系统、屏幕分享工具、协作文档创作、会议管理及在线游戏等。

（3）办公类软件：这类软件用于提高办公效率，如文字处理、制表、幻灯片编辑与播放工具，以及数据库程序等。基于SaaS云服务的办公软件具备协同的特征，便于分享，这是传统的本地化办公软件所没有的。

（4）软件工具类：这类软件用来解决安全性或兼容性问题，以及在线软件开发，如文档转换工具、安全扫描和分析工具、合规性检查工具及线上网页开发等。

随着因特网进一步延伸到世界各地，带宽和网速进一步改善，以及云服务提供商通过近距离部署分支云端，从而进一步降低网络延时，可以预计，能够云化的软件种类将越来越多，但是无论如何，下面三类软件不适合作为公共SaaS云服务出租。

（1）实时处理软件：比如飞行控制系统、工厂作业机器人控制等，这类软件要求任务完成时间非常精准（甚至达到微秒级）。假如云化，单单横亘在SaaS云端与消费者之间的网络延时就是一个难以承受的不可控因素，更不要说其他因素了。

（2）时时产生并处理大量消费者数据的软件：诸如视频监控、环境信息收集处理等时时产生并要处理大规模的数据信息（可能每秒达GB级）的软件，在目前和未来几年内的因特网宽带条件下，不适合云化，因为如此大规模的数据很难实时传输到SaaS云端进行处理。

（3）关键软件：这类软件如果运行异常，将会导致人员伤亡或者重大财产损失。为了提高这类软件的可靠性，最主要的方法是降低软件的复杂度——软件本身的复杂度和运行环境的复杂度，因为越简单就越可靠。而SaaS云服务环境包含着复杂的软/硬件栈(9层的IT系统)，以及不可预测的网络带宽、延时、丢包因素，绝对不合适云化此类关键软件并以公共SaaS云服务模式交

付给用户使用。

针对私有云或者社区云，如果云端就在本地，且消费者通过局域网接入云端，那么上面三类软件也可以 SaaS 模式部署。

与传统的软件运行模式相比，SaaS 模式具有如下优点：

（1）云终端少量安装或不用安装软件。直接通过浏览器访问云端 SaaS 软件，非常方便且具备很好的交互体验，消费者使用的终端设备上无须额外安装客户端软件。配置信息和业务数据没有存放在云终端里，所以不管用户何时何地使用何种终端操作云端的软件，都能看到一样的软件配置偏好和一致的业务数据。云终端成了无状态设备。

（2）有效使用软件许可证。软件许可证费用能大幅度降低，因为用户只用一个许可证就可以在不同的时间登录不同的计算机；而在非 SaaS 模式下，必须为不同的计算机购买不同的许可证（即使计算机没被使用）——过度配置许可证现象。另外，专门为保护软件产权而购置的证书管理服务器也不用买了，因为在 SaaS 模式下，软件只运行在云端，软件开发公司只跟云服务提供商打交道并进行软件买卖结算即可。

（3）数据安全性得到提高。对于公有云和云端托管别处的其他云来说，意味着 SaaS 型软件操纵的数据信息存储在云端的服务器中，云服务提供商也许把数据打散并把多份数据副本存储在多个服务器中，以便提高数据的完整性，但是从消费者的视角看，数据被集中存放和管理。这样做有一个明显的好处，那就是云服务提供商能提供专家管理团队和专业级的管理技术和设备，如合规性检查、安全扫描、异地备份和灾难恢复，甚至是建立跨城市双活数据中心。当今大的云服务提供商能够使数据安全性和应用软件可用性达到 4 个“9”的级别。对于云端就在本地的私有云和社区云来说，好处类似于公有云，但是抗风险能力要差一些，除非对大的意外事件提前做好预案，如为应对天灾（地震、洪水等）人祸（火灾等），建立异地灾备中心。另外，无处不在的网络接入，使人们再也不用复制数据并随身携带，从而避免数据介质丢

失或者被盗。数据集中存放和管理还有利于人们分享数据信息。

（4）有利于消费者摆脱 IT 运维的技术泥潭而专注于自己的核心业务。SaaS 云服务消费者只要租赁软件即可，而无须担心底层（基础设施层、平台软件层和应用软件层）的管理和运维。

（5）消费者能节约大量前期投资。消费者不用装修机房，不用建设计算机网络，不用购买服务器，也不用购买和安装各种操作系统和应用软件，这样就能节省成百上千万元的资金。众所周知，在非云计算模式下，这些巨额的前期投资在一到两年的时间内是不会产生任何效益的，因为一个中等规模的 IT 系统建设工期就要几年的时间——包括机房选址和装修，网络设备采购和综合布线，机器设备采购、安装和调试，软件部署、测试和转产上线等。

但是 SaaS 云服务也给人们带来了新的挑战，如完全依赖网络、跨因特网对安全防范措施要求更高、云端之间的数据移植性不够好、租户隔离和资源使用效率二者之间需要综合平衡考虑（详见第 5 章中“租户隔离”的相关内容）等。

下面是一些 SaaS 云服务的实际应用。

（1）电子邮件和在线办公软件：用于处理邮件、文字排版、电子表格和演示文档的应用软件，如谷歌邮箱（http://mail.google.com）、网易邮箱（http://mail.163.com/）、微软 Office 365 在线办公、谷歌在线文档（https://docs.google.com）等。

（2）计费开票软件：用于处理客户有关使用和订阅产品及服务所产生的账单。

（3）客户关系管理系统（CRM）：功能涵盖从客户呼叫中心到销售自动化的各种应用程序。

（4）协作工具：这种软件能促进企业内部或者跨企业的团队中的成员协同合作。

（5）内容管理系统（CMS）：用于管理数字内容，包括文本、图形图像、Web 页面、业务文档、数据库表单、视频、语音、XML 文件等，引入版本控制、权限管理、生命周期等。

（6）财务软件。

（7）人力资源管理系统。

（8）销售工具。

（9）社交网络：如微信、WhatsApp、LINE 等。

（10）企业资源计划（ERP）。

（11）谷歌在线翻译。

2.7　4 种部署模型

云计算有 4 种部署模型，分别是私有云、社区云、公有云和混合云，这是根据云计算服务的消费者来源划分的，即：

（1）如果一个云端的所有消费者只来自一个特定的单位组织（如微算科技公司），那么就是私有云。

（2）如果一个云端的所有消费者来自两个或两个以上特定的单位组织，那么就是社区云。

（3）如果一个云端的所有消费者来自社会公众，那么就是公有云。

（4）如果一个云端的资源来自两个或两个以上的云，那么就是混合云。目前绝大多数混合云由企事业单位主导，以私有云为主体，并融合部分公有云资源，也就是说，混合云的消费者主要来自一个或几个特定的单位组织。

2.7.1 私有云

私有云的核心特征是云端资源只供一个企事业单位内的员工使用，其他的人和机构都无权租赁并使用云端计算资源。至于云端部署何处、所有权归谁、由谁负责日常管理，并没有严格的规定。

1．云端部署何处

这有两个可能，一是部署在单位内部（如机房），称为本地私有云；二是托管在别处(如阿里云端)，称为托管私有云。本地私有云如图 2-21 所示。

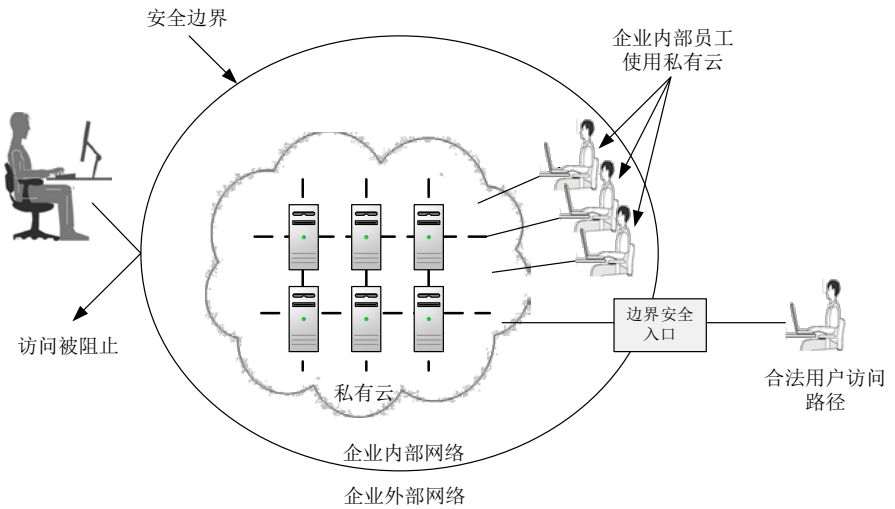


图 2-21　本地私有云

由于本地私有云的云端部署在企业内部，私有云的安全及网络安全边界定义都由企业自己实现并管理，一切由企业掌控，所以本地私有云适合运行企业中关键的应用。托管私有云是把云端托管在第三方机房或者其他云端，计算设备可以自己购买，也可以租用第三方云端的计算资源，消费者所在的企业一般通过专线与托管的云端建立连接，或者利用叠加网络技术在因特网上建立安全通道（VPN），以便降低专线费用，托管私有云如图 2-22 所示。

托管私有云由于云端托管在公司之外，企业自身不能完全控制其安全性，

所以要与信誉好、资金雄厚的托管方合作，这样的托管方抵御天灾人祸的能力更强。

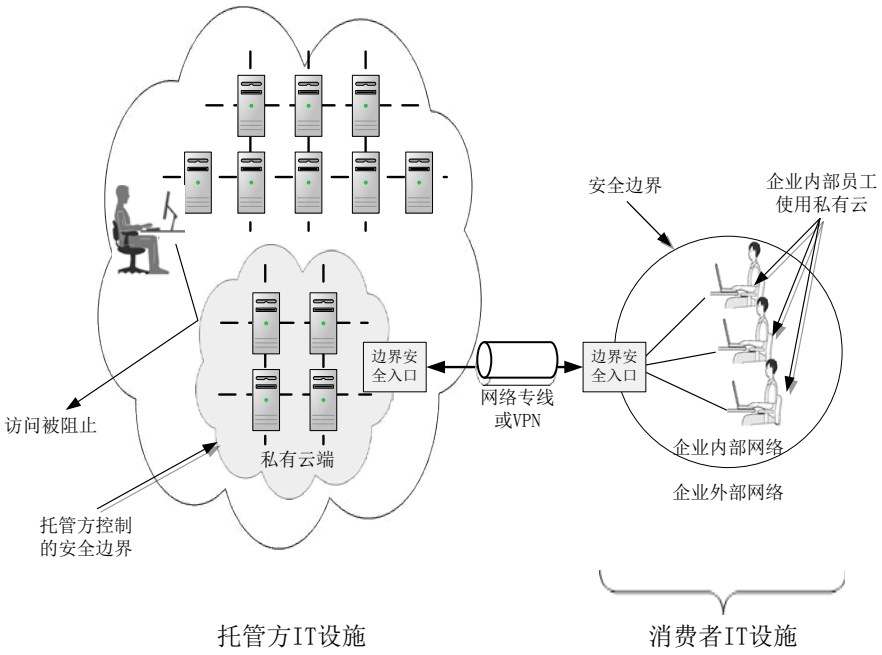


图 2-22　托管私有云

2．云端所有权归谁

云端所有权归谁，这存在两种可能，一种是归企业自身所有；另一种是归他人所有，企业租用。绝大多数本地私有云属于第一种情况，而对于托管私有云来说，租赁计算设备更具成本优势，云端规模伸缩也更自如。

3．云端由谁负责日常管理

一个云端的日常管理包括：管理、运维和操作。管理是指制定规章制度、合规性监督、定期安全检查、灾难演练、数据恢复演练、SLA 制订与落地检查等，侧重于制度和人员层面。运维指日常运行维护，具体包括机器性能监控、应用监控、性能调优、故障发现与处理、建立问题库、问题热线坐席、定期输出运维报告、产能扩容与收缩、应用转产与退出等，侧重于设备层面。

云端的操作不是指云服务消费者的操作，而是指云端的例行日常工作，包括数据备份、服务热线坐席、日常卫生、与消费者的一些操作互动等。云端的日常管理可以完全由自己承担，也可以完全外包出去或者部分外包出去。

私有云的规模可大可小，小的可能只有几个或者十几个用户，大的会有数万个甚至十几万个用户，但是过小的私有云不具备成本优势且计算资源配置的灵活性体现不出来，比如家庭和小微型企业，直接采用虚拟化即可，技术简单、管理方便。就像智能照明系统不适合三口之家的小居室一样，因为只有几盏灯、几个开关，手动操作简单、方便，成本也低。

企业私有办公云现在被很多大中型单位组织采用，用云终端替换传统的办公计算机，程序和数据全部放在云端，并为每个员工创建一个登录云端的账号，账号和员工一一对应，相比传统的计算机办公有如下好处。

（1）员工可在任何云终端登录并办公，可实现移动办公。

（2）有利于保护公司文档资料。

（3）维护方便。终端是纯硬件，不用维护，只要维护好云端即可。

（4）降低成本。购买费用低，使用成本低，终端使用寿命长，软件许可证费用降低。

（5）稳定性高。对云端集中监控和布防，更容易监控病毒、流氓软件和黑客入侵。

2.7.2 社区云

社区云的核心特征是云端资源只给两个或者两个以上的特定单位组织内的员工使用，除此之外的人和机构都无权租赁和使用云端计算资源。参与社区云的单位组织具有共同的要求，如云服务模式、安全级别等。具备业务相关性或者隶属关系的单位组织建设社区云的可能性更大一些，因为一方面能降低各自的费用，另一方面能共享信息。比如，深圳地区的酒店联盟组建酒店社区云，以满足数字化客房建设和酒店结算的需要；又比如，由一家大型

企业牵头，与其提供商共同组建社区云；再比如，由卫生部牵头，联合各家医院组建区域医疗社区云，各家医院通过社区云共享病例和各种检测化验数据，这能极大地降低患者的就医费用。

与私有云类似，社区云的云端也有两种部署方法，即本地部署和托管部署。由于存在多个单位组织，所以本地部署存在三种情况：一是只部署在一个单位组织内部；二是部署在部分单位组织内部；三是部署在全部单位组织内部。如果云端部署在多个单位组织内部，那么每个单位组织内部只部署云端的一部分，或者做灾备，本地社区云如图 2-23 所示。

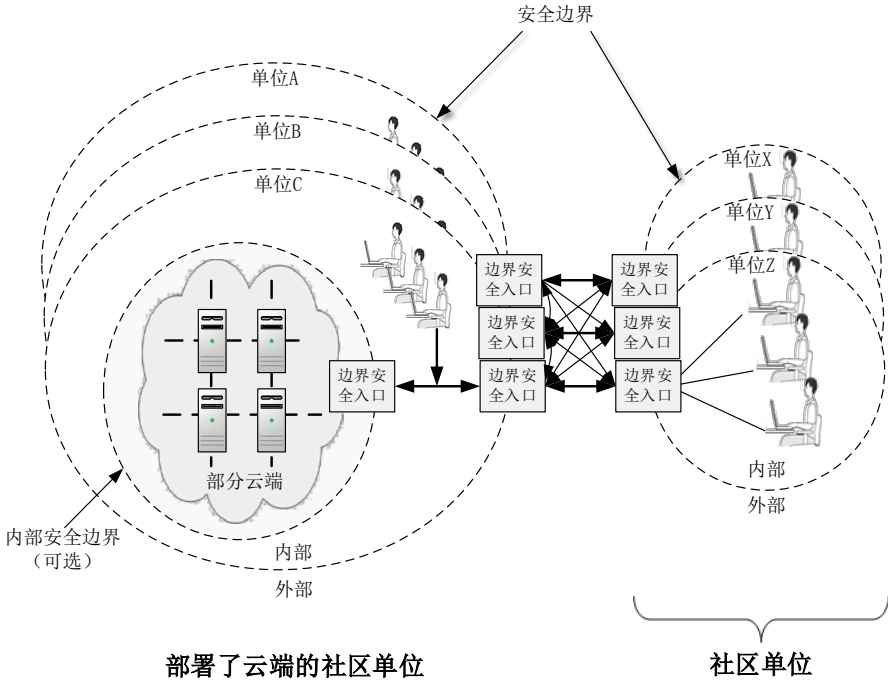


图 2-23　本地社区云

当云端分散在多个单位组织时，社区云的访问策略就变得很复杂。如果社区云有 *N* 个单位组织，那么对于一个部署了云端的单位组织来说，就存在 *N*–1 个其他单位组织如何共享本地云资源的问题。换言之，就是如何控制资源的访问权限问题，常用的解决办法有“用户通过诸如 XACML 标准自主访

问控制”、“遵循诸如‘基于角色的访问控制’安全模型”、“基于属性访问控制”等。除此之外，还必须统一用户身份管理，解决用户能否登录云端的问题。其实，以上两个问题就是常见的权限控制和身份验证问题，是大多数应用系统都会面临的问题。

托管社区云如图 2-24 所示。

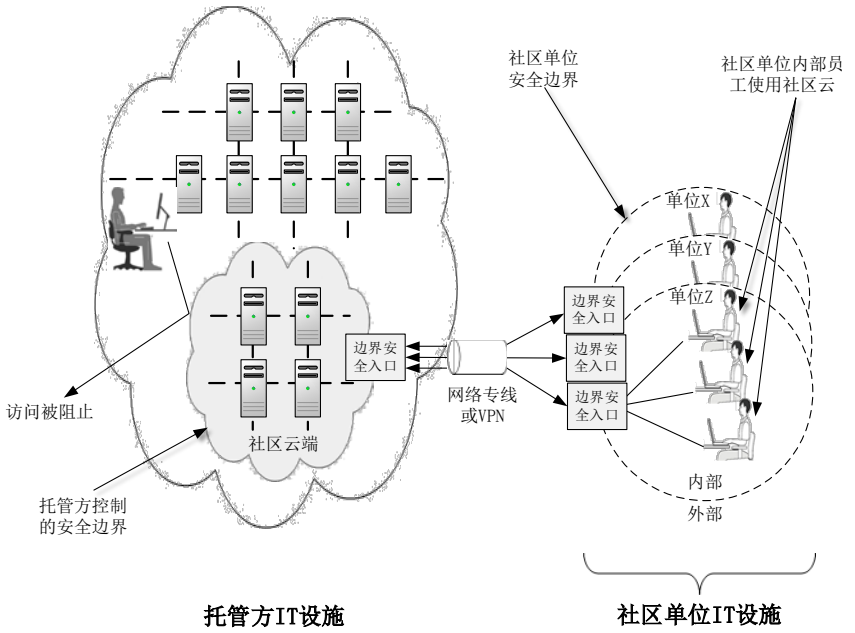


图 2-24　托管社区云

类似于托管私有云，托管社区云也是把云端部署到第三方，只不过用户来自多个单位组织，所以托管方还必须制订切实可行的共享策略。

2.7.3　公有云

公有云的核心特征是云端资源面向社会大众开放，符合条件的任何个人或者单位组织都可以租赁并使用云端资源。公有云的管理比私有云的管理要复杂得多，尤其是安全防范，要求更高。

公有云的一些例子：深圳超算中心、亚马逊、微软的 Azure、阿里云等。

2.7.4 混合云

混合云是由两个或两个以上不同类型的云（私有云、社区云、公有云）组成的，它其实不是一种特定类型的单一云，其对外呈现出来的计算资源来自两个或两个以上的云，只不过增加了一个混合云管理层。云服务消费者通过混合云管理层租赁和使用资源，感觉就像在使用同一个云端的资源，其实内部被混合云管理层路由到真实的云端了，混合云如图 2-25 所示。

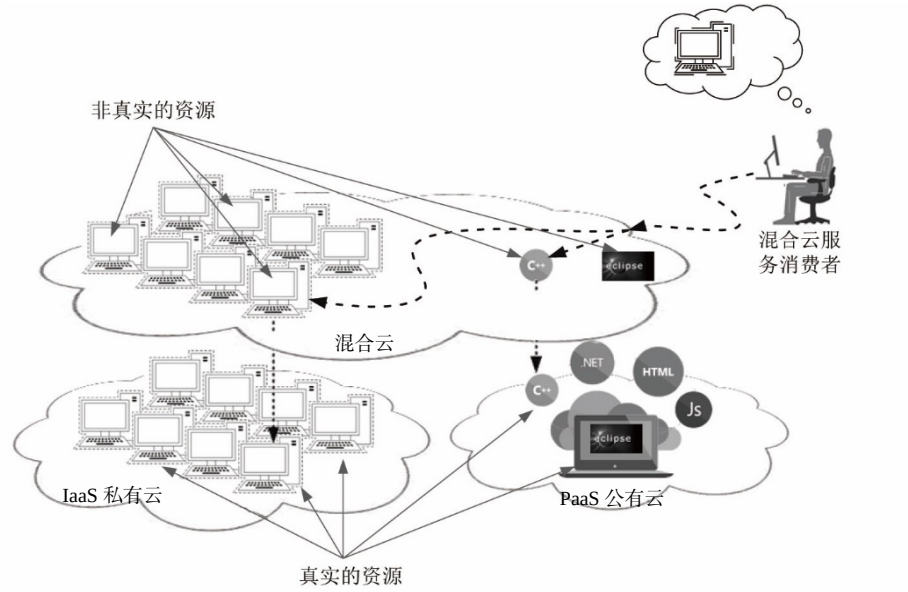


图 2-25 混合云

在图 2-25 中，假如用户在混合云上租赁了一台虚拟机（IaaS 型资源）及开发工具（PaaS 型资源），那么用户每次都是连接混合云端，并使用其中的资源。用户并不知道自己的虚拟机实际上位于另一个 IaaS 私有云端，而开发工具又在另一个公有云上。

由于私有云和社区云具有本地和托管两种类型，再加上公有云，共有 5 种类型，所以混合云的组合方式就有很多种形式了，混合云的组合方式如图 2-26 所示。

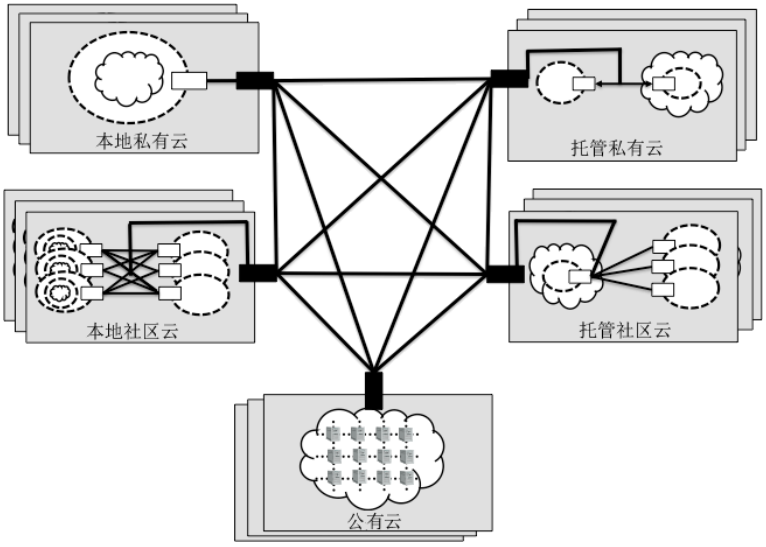


图 2-26　混合云的组合方式

混合云属于多云这个大类，是多云大类中最主要的形式，而公/私混合云又是混合云中最主要的形式，因为它同时具备了公有云的资源规模和私有云的安全特征。从图 2-27 中可以看出，私有云和公有云构成的混合云占比达到 55%。下面将围绕这种形式的混合云来展开阐述。

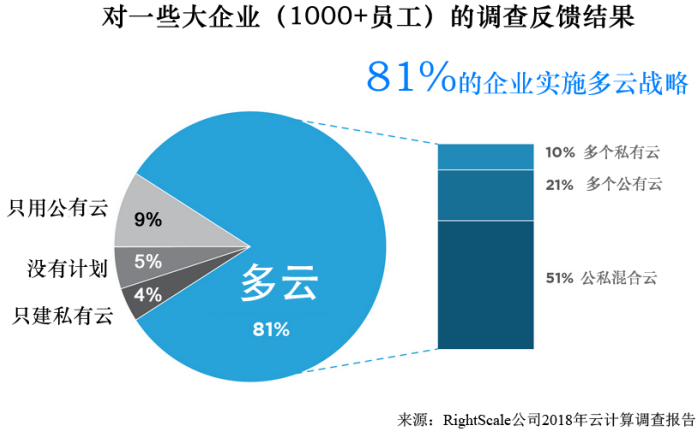


图 2-27　RightScale 公司给出的 2018 年报告

1．公/私混合云的优势

（1）架构更灵活：可以根据负载的重要性灵活分配最适合的资源，例如将内部重要数据保存在本地云端，而把非机密功能移动到公有云区域。

（2）技术方面更容易掌控。

（3）更安全：具备私有云的保密性，同时具有公有云的抗灾性（在公有云上建立虚拟的应急灾备中心或者静态数据备份点）。

（4）更容易满足合规性要求：云计算审计员对多租户的审查比较严格，他们往往要求云计算服务提供商必须为云端的某些（或者全部）基础设施提供专门的解决方案。而这种混合云由于融合了专门的硬件设备，提高了网络安全性，更容易通过审计员的合规性检查。

（5）更低的费用：租用第三方资源来平抑短时间内的季节性资源需求峰值，相比自己配置最大化资源以满足需求峰值的成本，这种短暂租赁的费用要低得多，平抑季节性资源需求峰值如图 2-28 所示。

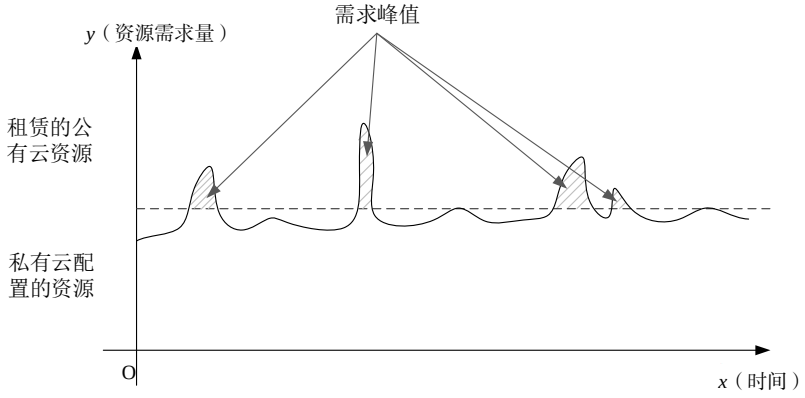


图 2-28　平抑季节性资源需求峰值

2．公/私混合云的构成

（1）私有云：这是混合云的主要组成部分，企业部署混合云的步骤一般是先“私”后“公”。

（2）公有云。

（3）公/私云之间的网络连接：一般为公有云提供商提供的高速专线，或者是第三方的 VPN。

（4）混合云管理平台：是用户的统一接入点，实现资源的自动化、费用结算、报表生成、云端日常操作及 API 调用等，可以进一步细化为数据管理、虚拟机管理、应用管理等几个层面的软件。目前混合云管理平台产品有 VMware vRealize、Microsoft System Center、RightScale 等。

3．公/私混合云的功能

公/私混合云可以做多个层面的事情，基本可以分为数据层面和业务负载层面。在这两个层面，有一些典型的应用场景，具体如下。

（1）数据备份：将私有云的数据备份到更便宜和可靠的公有云上。

（2）灾备：在私有云出现故障时，由公有云上的灾备环境提供服务。

（3）负载延伸：当私有云无法提供新增负载所需要的资源时，在公有云上创建虚拟主机来支持新的负载，当负载下降后再删除这些虚拟主机回到纯私有云。

（4）使用公有云作为开发测试云。

2.8 云计算的优劣分析

2.8.1 情景案例

云计算与传统的计算机系统相比，具有明显的优势，为了描述清楚这种优势，请看下面的情景案例。

我是 XYZ 公司的李老板，员工人数在 20 人以上，其中 2/3 的人需要用计算机办公，公司会用到下面的软件与系统。

（1）Word/Excel/PowerPoint：用于处理文字材料、电子表格和制作并演示PPT给客户观看。需要购买微软或者金山的WPS办公软件。

（2）办公自动化软件：用于公司内部语音通话、视频会议、消息通信、审批自动流转、文件转发、收发传真等。

（3）建立公司的网站：比如网站域名是 www.xyz.com，上下游公司都可以通过公司的网站了解我们的业务、反馈意见和建议，公司的重大新闻也要发布到网站上。公司网站就是公司的窗口和门户，一定要设计得专业美观，而且紧扣公司的主营业务。

（4）公司邮件系统：我们要为每个员工分配一个公司邮箱地址，类似于xxxx@xyz.com格式，邮箱地址后缀统一为公司的域名，然后将邮箱地址印在员工的名片上。名片上绝对不能印免费的邮箱地址，因为如果一家公司连自己的邮件系统都没有，那么绝对是一家小公司。

（5）ERP 系统：主要用来管理进、销、存和生产、财务、人事等，打通各个部门的业务数据通道，引入一系列的业务流程，最终目的是降低库存，留住和挖掘客户资源，加快资金周转，减少人力成本。ERP 系统安装复杂、价格昂贵、日常管理工作量大。公司从创立之初就开始使用ERP比公司做大后再使用ERP效果更好，等公司做大后再使用ERP很容易失败。

（6）产品数据管理软件：公司的产品线多，涉及成千上万的零部件，而且每个零部件又有几十个版本，所以我们不得不采用专门的产品数据管理软件来管理大量的产品数据。自从采用产品数据管理软件以来，产品研发周期明显缩短，而且版本控制有条不紊。

（7）AutoCAD/Photoshop/Solidworks/Cadence：我们必须要用这些专业的产品设计工具，而正版软件的价格异常昂贵，每年还有升级费用；公司做大后，不敢使用盗版软件，否则法律风险很高。

（8）产品可靠性工程管理软件：为了提高公司产品的可靠性，我们必须要使用这种软件，但这样的软件价格都在百万元以上。

另外，还有如下要求。

（1）公司内部的资料在没有授权的情况下，员工不能带出公司。必须严格保护公司的知识资产，包括各种文档资料、图纸、源代码、产品数据等。

（2）员工出差时也可以随时访问公司内部的 ERP 等系统，即做到移动办公。

（3）要求公司的网站、邮件系统和 ERP 系统不关机，也就是要每天 24 小时运行，允许员工和客户随时访问。

（4）严格控制购买计算机和软件的成本，以及日常的运行维护成本，包括电费和计算机工程师的人力成本，资金都要用在刀刃上。

（5）要求采用最先进的软件，如产品设计软件、ERP 等。

为了满足我的需求，我会参照如下方式行事，公司规模不同方案也不同。

1．假如我的公司是一家小型公司，员工人数在 200 人以内

（1）部署远程桌面服务办公环境：购买几台服务器部署微软的远程桌面服务，每个办公桌上放置一台云终端，给需要的员工每人一个账号和密码。各种软件（如办公软件、产品设计软件等）都安装在服务器上并且在服务器上运行，公司全部的文档资料也放在服务器上。云终端是纯硬件设备，里面不用安装 Windows 系统和各种应用软件。

这样做的好处有：

- 购买计算机设备的成本低很多。
- 只要购买正版软件，节约的软件许可证费用非常可观。
- 终端折旧周期长（8 年以上），耗电极低，不容易出故障。
- 数据资料无法复制，云终端的 U 盘插口数据只能复制进去不能复制出来，数据资料集中存放在服务器上。
- 便于移动办公，员工可以在任何一台终端上登录云计算中心并办公。
- 计算机的日常维护工作量小，只要维护好服务器即可。

- 不容易感染病毒。
- 出差在外也可以轻松访问云端，实现移动办公。

（2）租用公有云上的 ERP 软件、产品数据管理软件和可靠性工程软件等，前提是有这些软件的 SaaS 云服务提供商。这些软件价格昂贵，如果单独购买安装在公司内部，那么对于小公司来说，是没有这个资金实力的，但是我们可以租用使用权。按账号每月付费，我租了 10 个 ERP 账号、3 个产品数据管理系统的账号和 2 个可靠性工程软件账号，先付了一年的租金，总金额不到 3 万元。员工使用租来的账号和密码登录公有云后，就可以使用公有云中的软件，数据也存放在其中。公有云从不关机，所以我们可以随时随地访问 ERP 系统、产品数据管理系统和可靠性工程软件系统。

这样做的好处有：

- 小公司也可以使用以前只有大公司才用得起的大型软件系统，从而提高小公司的竞争力。
- 无须费心去日常管理这些系统。
- 无须购买服务器来安装这些系统，省去了不少电费。

（3）租用公有云上的一台虚拟机专门运行公司网站和邮件系统，每年租金人民币 2000 元。虚拟机从不关机，满足了我们的要求：网站和邮件系统随时可用。

如果我的公司之前用普通计算机办公，那么我就逐渐实施私有办公云，等计算机淘汰后就替换成云终端。这样，公司的计算机随着时间的推移会越来越少，而云终端会越来越多。

对于一家员工人数在 200 ~ 500 人的中型公司，方案与上面的大致相同，只不过要购买更多的服务器，租用更多的公有云账号。

2．假如我的公司是一家大型公司，员工人数在 500 人以上

部署本地私有云，全部服务器池化——10%的服务器最终以物理机的形式

让用户租用，90%的服务器最终以虚拟机的形式让用户租用。因为公司一些核心的应用必须运行在物理机上，以便满足安全性要求。云端设计成可伸缩的，服务器随着办公人数的变化而睡眠或者唤醒，比如晚上加班的人数少，大部分服务器处于睡眠状态，早上随着上班人数的不断增加，更多的服务器被不断唤醒。另外，还要考虑一定数目的备份服务器，允许坏 3 台服务器而不会影响业务。还要对服务器做集群划分，不同的部门使用不同的服务集群，从而轻松实现安全控制。

（1）员工办公使用的资源有两类，一是员工独占一台虚拟机，二是多个员工共享一台虚拟机（远程桌面服务模式）。对于一个具体的员工来说，到底是独占虚拟机还是与他人共享，要根据公司的安全策略来定。

这样做的好处是：

- 公司的各种文档资料能得到很好的保全，包括产品图纸、源代码、合同文本、客户资料等。就像波音公司那样，产品设计工程师全部采用云终端来完成产品设计，图纸是复制不走的。
- 极大地降低了对 IT 的投入，包括硬件的采购成本、软件的采购和升级费用、日常运行维护成本和计算机工程师的人力成本。
- 极大地提高了电子化办公的可靠性和稳定性。传统的采用台式机办公普遍存在各种不稳定的因素，如病毒入侵、不正常关机导致计算机软件破坏、计算机硬件故障、软件安装配置不正常、数据丢失等；而采用私有办公云，这一切问题都不复存在了。
- 安装和升级软件极其方便，只要在服务器上操作即可，不涉及众多的云终端。要知道，大公司会使用数百个各种各样的软件，如果这些软件都要安装在每个员工的计算机上，那么工作量可想而知。
- 便于做安全控制，如局域网接入认证、用户上网行为控制、日志登记、员工桌面监控、外发邮件监控、病毒查杀和入侵检测等。
- 实现移动办公，员工可以在公司内部的任何一台云终端上使用自己的账号登录云计算中心办公，员工与计算机不再一一绑定。尤其是跨地区的

集团公司，移动办公更能体现其优势。通过配置 VPN 接入，轻松实现出差在外的员工登录公司内部的云计算中心。

- IT 日常运维工作变得异常简单，只要维护好云计算中心即可，从而可以减少大量的计算机运维工程师。

（2）在云端专门划分一个集群，在该集群上的虚拟机中运行公司的 ERP 系统、产品数据管理系统、网站系统、邮件系统，以及其他大型应用系统。利用虚拟机做高可靠性比物理机更容易，因为虚拟机能轻松实现热迁移。在运行能力富余的情况下，公司还可以对外销售 SaaS 云服务，让其他中小型公司租用这些昂贵的软件系统。

这样公司能灵活控制这些系统，方便积淀各种数据并做大数据分析，对外出租 SaaS 账号相比纯粹的公有云运营商更贴近用户的需求，因为我们本身就在使用这些系统。

2.8.2 云计算的优势

1．对于社会

（1）降低全社会的 IT 能耗，减少排放，真正做到“绿色计算”。

（2）提高全社会的 IT 设备使用率，并降低电子产品的数量，从而减少因设备淘汰而产生的电子产品垃圾，对于保护环境大有裨益。

（3）信息技术产业进一步合理分工——由资金雄厚、技术过硬、专业人士众多的机构负责建设并管理云端，从而提高了整个社会信息技术处理环境的可靠性。换言之，也就降低了因天灾人祸导致的生命财产损失。

（4）形成新的云计算产业。

（5）有利于全社会共享数据信息，打破信息孤岛。尤其是涉及公民的身份信息、档案信息、信用信息、健康信息及教育工作信息等的全国性公有云平台，带来的社会效益更是巨大的。

2．对于云计算消费者

（1）降低了信息技术成本：前期投入和日常使用成本得到大幅度降低，同时也降低了因各种 IT 事故导致的损失。

（2）提高了数据的安全性，具体介绍见后续章节。

（3）提高了应用系统的可靠性，具体介绍见后续章节。

（4）提高了用户体验：当今网络无处不在，云计算消费者可以随时随地采用任何云终端接入云端并使用云中的计算资源，真正实现移动办公。

（5）大型昂贵软件平民化：诸如可靠性工程软件、ERP 系统、CRM 系统、商业智能系统等云化之后以 SaaS 模式出租，这些以前只有大型企业使用的软件系统，现在广大中小型企业和个人都能用得起。

（6）从复杂的 IT 技术泥潭中摆脱出来，专注于自己的核心业务和市场。

（7）能快速响应消费者对计算资源的弹性需求，从而能及时满足企业的业务变化。在传统 IT 系统下，一项新业务对 IT 资源的扩容要求，往往在数月或者一年后才能得到满足，这使得市场人员和管理层往往难以接受，因为市场是瞬息万变的。

（8）有利于企业之间或者个人之间共享信息，打破信息孤岛。

（9）个人、中小企业和机构也用得起高性能计算。

2.8.3 云计算的劣势

（1）严重依赖网络。在没有网络的地方，或者在网络不稳定的地方，消费者可能根本无法使用云服务或用户体验很差。但这并不是云计算固有的缺陷，随着网络普及越来越广、网速越来越快，甚至是城市无线 Wi-Fi 全覆盖、国家无线 Wi-Fi 全覆盖的到来，将使网络不再是问题。针对这个问题，现在有一些胖云终端产品，它会把一些常见的应用程序驻留在本地，同时缓存数据，当网络良好时，数据自动与云端同步。

（2）可能导致数据泄密的环节增多。云端、灾备中心、离线备份介质、网络、云终端、账号和密码等，这些都有可能成为信息的泄密点。但是云计算使数据信息遭到非人为因素破坏的概率大大降低了，比如在传统 IT 系统中，存储设备损坏、机房火灾、地震、雷电、洪水等都会破坏数据，而在云计算环境则没有这些隐患。总之，云计算消除了一些数据泄密和破坏点，但同时又带来了一些新的不安全因素。

（3）相对于传统的分散计算，云计算把计算资源集中在一起，因而风险也被集中在一起。云端成了单点故障，如果云端发生事故，则影响面将非常巨大。目前常见的应对措施是数据冗余存储、建立灾备中心、建立双活数据中心等。

（4）用户对数据和技术的掌控灵活度下降。对于 IaaS 云服务，用户无法掌控基础设施层；对于 PaaS 云服务，用户无法掌控基础设施层和平台软件层；而对于 SaaS 云服务，用户失去了基础设施层、平台软件层和应用软件层的掌控。另外，数据存放在云端，如果数据量巨大，那么用户移动数据耗时又耗力，如果网速慢，则势必会严重影响数据掌控的灵活性。不过，对技术掌控降低反过来表示用户可以脱离繁杂的技术陷阱，从而专心关注企业的核心业务和市场，因此这也是优势。

第 3 章

“云”架构

本章讲述云架构的参考模型。

3.1　云架构的参考模型

图 3-1 所示是美国国家标准与技术研究所（NIST）定义的通用云计算架构参考模型，图中列举了主要的云计算参与者，以及他们各自的分工。

NIST 云计算架构参考模型定义了 5 种角色，分别是云服务消费者、云服务提供商、云服务代理商、云计算审计员和云服务承运商。每个角色可以是个人，也可以是单位组织。如表 3-1 所示，列举了每个角色的具体定义。

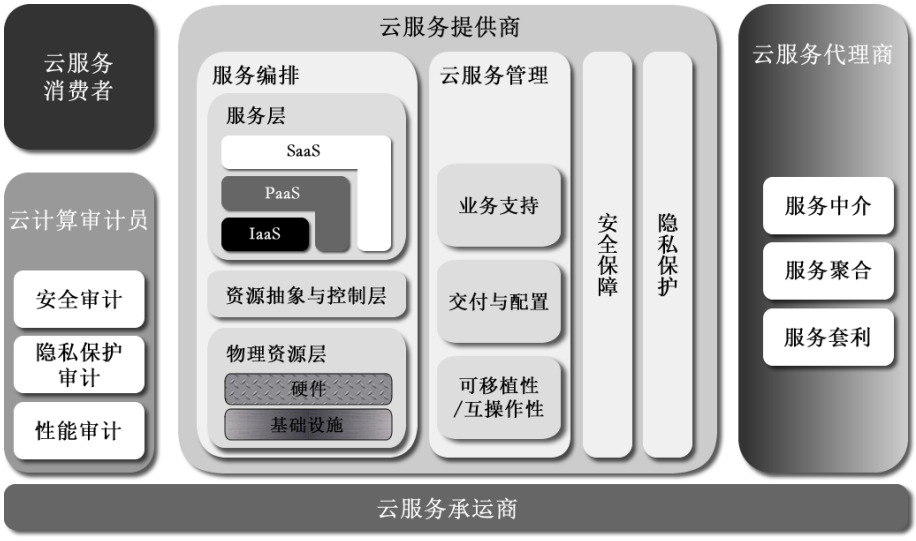


图 3-1　云计算架构参考模型

表 3-1　角色定义

角　色	定　义
云服务消费者	租赁云服务产品的个人或者单位组织
云服务提供商	提供云服务产品的个人或者单位组织
云服务代理商	代理云服务提供商向消费者销售云计算服务并获取一定佣金的个人或者单位组织，如世纪互联是微软 Azure 的中国代理商
云计算审计员	能对云计算安全性、云计算性能、云服务及信息系统的操作开展独立评估的第三方个人或者单位组织
云服务承运商	在云服务提供商和云服务消费者之间提供连接媒介，以便把云计算服务产品从云服务提供商那里转移到云服务消费者手中，如中国电信。但是广域网商和因特网商不属于云服务承运商

云计算中各个角色之间的交互如图 3-2 所示，云服务消费者可以从云服务代理商或者云服务提供商那里租赁云服务产品，而云计算审计员必须能从云服务消费者、云服务提供商和云服务代理商那里获取信息，以便独立开展审计工作。

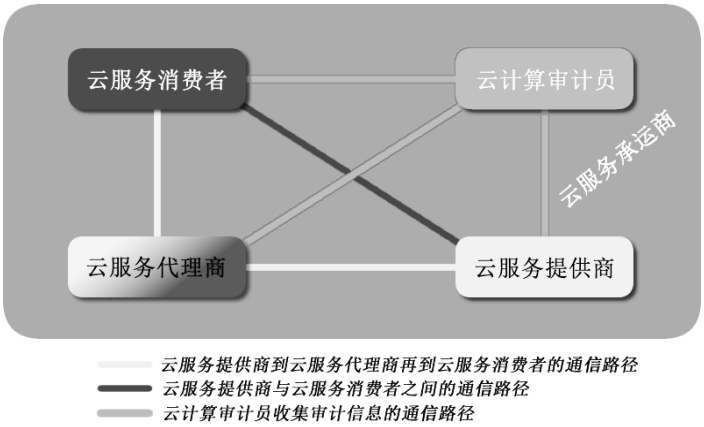


图 3-2　云计算中各个角色之间的交互

在具体的实施过程中，并不是每个云计算都包含这 5 种角色，但是云服务提供商和云服务消费者是必需的两个角色，而是否包含其他 3 种角色，与具体的业务要求相关。下面是一些例子。

例 1：只有云服务消费者和云服务提供商，如图 3-3 所示。

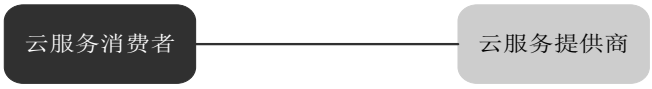


图 3-3　只有云服务消费者和云服务提供商

例如本地私有云、Dropbox 网盘等。

例 2：云计算包含云服务消费者、云服务提供商和云服务承运商，如图 3-4 所示。

图 3-4　包含云服务消费者、云服务提供商和云服务承运商

例如，XYZ 公司把数据备份到亚马逊公有云上，那么 XYZ 公司就是云服务消费者，亚马逊是云服务提供者，而提供点对点专线服务的本地电信部门就是云服务承运商。

例 3：云计算包含云服务消费者、云服务代理商和多个云服务提供商，如图 3-5 所示。

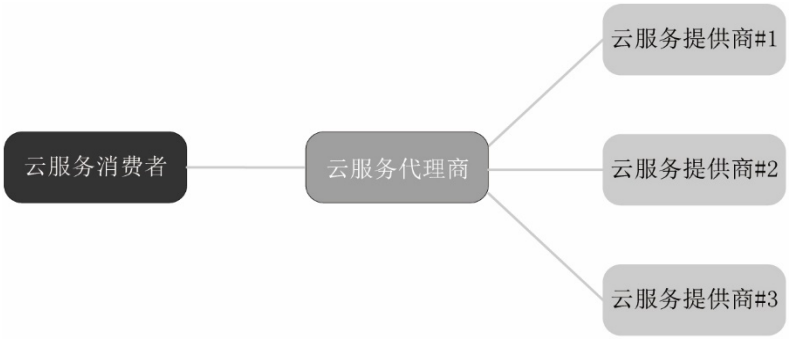


图 3-5　包含云服务消费者、云服务代理商和多个云服务提供商

例如，一家私有企业 XXX 利用第三方的云计算代理咨询公司发现并租赁最经济的 Linux 云主机，那么云服务消费者是 XXX 企业，云服务代理商是第三方的云计算代理咨询公司，云服务提供商是亚马逊、谷歌、阿里巴巴。在本例中，云服务消费者只与云服务代理商打交道，其并不知道真正的云服务提供商是谁。

例 4：云计算包含云服务消费者、云服务提供商和云计算审计员，如图 3-6 所示。

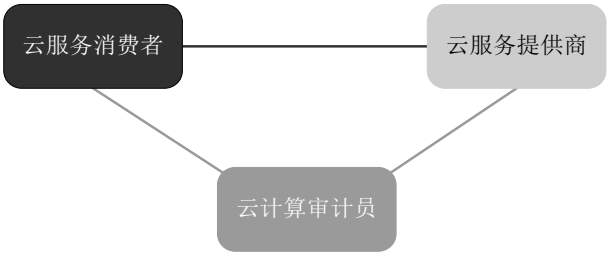


图 3-6　包含云服务消费者、云服务提供商和云计算审计员

例如，云服务消费者使用公民档案 SaaS 云服务，那么社会公民是云服务消费者，公民档案 SaaS 云服务提供商是云服务提供商，而人社部和云计算国家安全管理局是云计算审计员。

3.2 云服务消费者

云服务消费者从云服务提供商或者云服务代理商那里租赁云服务产品，在合同期内和云服务提供商保持一种供/需业务关系。首先准消费者浏览云服务提供商的官方网站，在云服务分类中找到能够满足自己需求的云服务产品，然后和云服务提供商签订合同，最后按照合同条款使用云计算资源并付款。付款的方式有很多种，如按年、按季、按月付款及按次数付款等。

需要注意的是，云服务消费者应与云服务提供商仔细落实服务水平协议（SLAs）中的条款，这些条款涵盖服务质量、安全保障、应用中断的弥补措施等内容。另外，云服务提供商可能在 SLAs 中列举针对所有用户的承诺条款，以及云服务消费者必须接受的限制和义务条款。通常一个云服务提供商的报价和 SLAs 条款是不可以讨价还价的，除非你是一个令他们心动的大客户，一般客户只有在货比三家后挑选信誉良好、财力技术雄厚、报价实惠、服务条款更具吸引力的云服务提供商。

不同云服务类型的消费者使用云服务产品的活动和流程也有所区别，如图 3-7 所示，描绘了不同类型的消费者可能使用到的云服务产品。

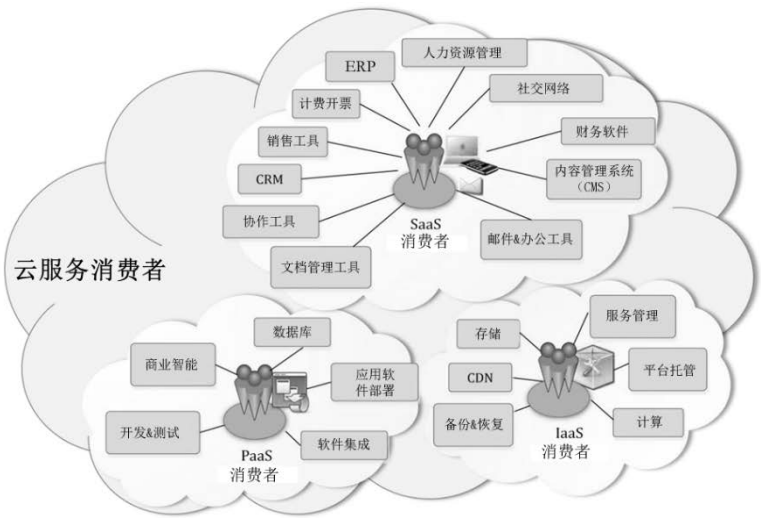


图 3-7　不同类型的消费者可能使用到的云服务产品

SaaS 型消费者通过网络访问云端的 SaaS 型应用程序。如果消费者是一个单位组织，那么它的内部成员才是真正的 SaaS 程序操作者，不过有的成员是使用软件的最终用户，有的成员是为最终用户配置软件的管理员。SaaS 型消费者付款的计算因素包括最终用户数、使用时间的长短、消耗的网络带宽、存储的数据规模或者数据保存期限等。

PaaS 型消费者使用云端的工具和计算资源来开发、测试、部署和管理驻留于云中的应用程序，因此消费者的具体角色可能是程序开发人员、软件测试工程师、应用程序部署人员或者应用程序管理员等。与 PaaS 型消费者付款有关的因素包括处理器消耗、网络带宽消耗、数据库存储规模及使用时间的长短等。

IaaS 型消费者使用云端的虚拟机、NAS 存储、网络设施及其他能部署和运行任何软件的基础计算资源，因此 IaaS 消费者一般为系统开发人员、系统管理员或者热衷于掌控基础设施的 IT 部经理。计费的因素包括虚拟机使用 CPU 的小时数、存储的数据规模和时间、消耗的网络带宽，以及申请的公网 IP 地址的数量等。

3.3 云服务提供商

云服务提供商是个人或者单位组织，他们负责搭建云端并对外提供云服务产品。支撑云服务的 IT 层可以由云服务提供商自己组建，也可以租赁其他人的云服务。比如，一个云服务提供商租赁其他人的虚拟机（IaaS 云服务），并在此虚拟机上部署 PaaS 云服务，然后再出租出去。

云服务提供商的五大任务包括服务部署、服务编排、云服务管理、安全保障和隐私保护，如图 3-8 所示。

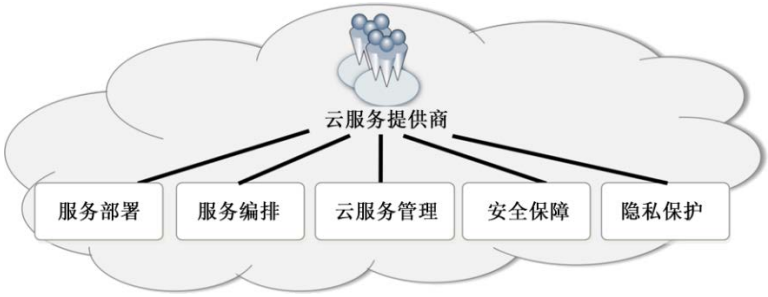


图 3-8　云服务提供商的五大任务

其中，服务部署任务参见第 2 章中的“2.7　4 种部署模型”部分，下面重点介绍其余四大任务。

3.3.1　服务编排

服务编排是指把各种系统组件合理地整合在一起，以便向云服务消费者提供云服务产品。

图 3-9 所示是从云计算架构参考模型中单独截取出来的云服务编排模型，全部的系统组件分为 3 种类型，分别对应云服务编排模型中的 3 个叠加层，从下至上依次是物理资源层、资源抽象与控制层、服务层。

图 3-9　从云计算架构参考模型中单独截取的云服务编排模型

顶部的服务层定义了云服务消费者访问计算资源的接口（简称访问接口）——IaaS、PaaS 和 SaaS。SaaS 应用程序可以（但不是必须）搭建在 PaaS 组件上，PaaS 组件可以（但不是必须）搭建在 IaaS 组件上，这三者层叠在一起是组合方式的一种。另外，从图 3-9 中可以看出，PaaS 和 SaaS 存在直角的单腿直立部分，表明可以单独构建各自的底层架构，比如 SaaS 云服务直接部署在物理服务器上，而不是层叠在 PaaS 和 IaaS 组件上。对于一个具体的云服务提供商而言，其可定义全部的（3 种）服务模式或者其中的一到两种。

中间层是资源抽象与控制层，它包括两类系统组件：一类组件是抽象组件，主要用于对物理计算资源进行虚拟化，虚拟化之后的资源就可以放入资源池中，以便将来供云服务消费者弹性租赁。抽象组件包括虚拟机管理平台（Hypervisor）、虚拟机软件（VM）、软件定义存储（SDS）、软件定义网络（SDN）等，其中虚拟机技术是最常见的资源抽象组件。无论如何，资源抽象组件必须能确保高效、安全和可靠地运用底层的物理资源。本层的另一类组件是控制组件，主要负责资源分配、访问控制和监控资源使用等。总之，本层的主要功能是把物理资源池化并有效管理被池化后的资源。

底层是物理资源层，包含所有的物理计算资源——物理服务器（主要包含 CPU、内存计算资源）、网络设备（路由器、防火墙、交换机、网卡等）、存储设备（如硬盘、存储阵列），以及其他物理计算设施；也包含基础设施资源，如加热器、空调、通风设备、供电设备等。

三层叠加在一起，上层依赖直接下层。资源抽象与控制层利用下层的物理资源，向上层暴露资源池，而顶部的服务层利用下层的资源池向云服务消费者暴露服务接口，而云服务消费者不能直接操纵底层的物理资源。

3.3.2 云服务管理

云服务管理是指云服务提供商必须履行的一套流程和任务，以便圆满地把云服务交付给消费者。这些流程和任务分为三大类：业务支持、交付与配置、可移植性与互操作性，云服务管理如图 3-10 所示。

图 3-10　云服务管理

1．业务支持

业务支持是与客户有关的活动和服务，具体包括如下内容。

（1）租户管理：管理租户账号、激活/禁用/终止账号、管理用户资料、解决租户提出的问题、处理投诉等。

（2）合同管理：管理服务合同、洽谈/签订/关闭/终止合同等。

（3）云服务产品管理：建立和管理云服务产品目录，以便用户浏览和选择等。

（4）费用结算：生成发票并送交客户，回收款项，尽量做到现收现付。

（5）报表与审计：监视用户消耗资源状况，产生有关报表，履行事后监督和审计。

（6）价格管理：给云服务产品建立价格点和价格分层，监视竞争对手的报价并及时调整产品价格以确保竞争力。云服务提供商通常会给大客户提供价格折扣或者信用消费（先消费后付款）。

2．交付与配置

交付与配置是与资源有关的管理活动，具体包括如下内容。

（1）快速交付：能快速响应用户的弹性需求，最好是自动化响应。

（2）资源变更：包括替换故障设备、升级设备、添加新设备，从而扩充资源容量，以及重新配置现有的设备。

（3）资源监控：发现和监视虚拟资源，监视云端操作和事件，并且产生性能报表。

（4）计量：定义付费的计量方法，比如根据某类资源（如存储设备、CPU、内存、网络带宽等）的使用量和时间长短计费，也有按照每使用一次计费的，还有的使用较复杂的计费公式。

（5）服务水平协议（SLA）管理：具体涵盖 SLA 定义、SLA 实施监督和 SLA 执行评价，目的是保证服务质量。

3．可移植性与互操作性

云计算能节约成本，快速满足用户对资源的弹性需求，这使得一些潜在的云服务消费者有兴趣把计算迁入云端。然而，这些潜在的云服务消费者能否变成真实的云服务消费者，在很大程度上依赖于云服务提供者如何对待用户关心的安全、可移植性和互操作性问题。

关于可移植性，潜在的消费者非常想知道他们是否能够以较低的成本和最小的中断时间在多个云端之间迁移数据或应用程序。而对于互操作性，用户关心在两个或多个云端之间的互通能力。

为了便于用户进行数据移动、服务交互和系统迁移，云服务提供者应该提供一些有利机制（包括策略和工具），而不是故意设置障碍，以便增加用户黏度。数据移动是指把数据复制到云端或从云端复制出来或者进行批量传输；理想的服务交互能力是指用户能够通过统一的管理接口使用横跨于多个

云端的数据和服务功能；而系统迁移是指将一台完全关闭的虚拟机从一个云服务提供商的云端迁移到另一个云服务提供商的云端，或者在不同云服务提供商的云端之间迁移应用、服务和相应的内容。

但是要特别注意的是，不同的云服务模式（IaaS、PaaS、SaaS）在可移植性和互操作性方面的侧重点不同。例如，IaaS 模式侧重于迁移虚拟机并能在新的云端启动。因此，为了能迁移一台虚拟机镜像到另一个采用不同虚拟机技术的云端，必须移除云服务提供商加入虚拟机镜像文件中的特有的扩展技术。如果这种扩展技术非常好且被广泛接受，那么云服务提供商应该公开这种技术细节，从而使得所有的 IaaS 云服务提供商都接受并把该技术融入自己的虚拟机镜像中。而 SaaS 侧重于数据可移植性方面，因此最好按照业界通行的标准进行数据抽取和备份，以便保证数据的可移植性。

3.3.3 安全保障

我们必须清晰地认识到安全性问题涉及云计算参考模型的各个方面，纵向上从底层的基础设施到顶层的应用，横向上包括全部的参与角色，如云服务提供商、云服务消费者、云服务代理商等。基于云的系统仍然需要满足这些传统 IT 系统中的安全要求，如认证、授权、可用性、保密性、身份管理、完整性、审计、安全监控、事件响应及安全策略管理等，对这些技术展开详细论述超出了本书的范围，下面着重讨论一下云计算特有的一些安全问题。

1．3 种云服务模式蕴含的安全问题

云计算的 3 种云服务模式 Iaas、PaaS 和 SaaS 为用户提供了不同类型的接入云端的接口，同时也不可避免地成为不怀好意者的攻击入口。因此，在设计和实施云计算的过程中，需要充分考虑这 3 种云服务模式的内部机制和各自侧重的安全性问题。例如，人们通常使用网页浏览器通过因特网访问云端的 SaaS 应用，那么在考察 SaaS 的安全性时，就需要重点关注网页浏览器的

安全性。而 IaaS 消费者一般是远程登录并使用云端的虚拟机，那么在设计 IaaS 云服务时，就要重点关注虚拟机软件的隔离效果，因为运行在同一台物理机上的虚拟机相互之间的隔离效果越好就越安全。

2．4 种部署模型蕴含的安全问题

4 种部署模型也蕴含着安全问题，其中从租户的隔离程度来考察部署模型中的安全问题是一个不错的方法。由于私有云只供一个单位组织内部的员工使用，所以在租户隔离方面就没有公有云那么重要。公有云上的租户来自五湖四海，身份复杂且不可预测，所以公有云上的租户必须严格隔离。另一个观察部署模型所蕴含的安全问题的解决方法是采用安全边界（在 2.7 节中介绍过），例如一个本地私有云的云端就不必再额外增加一道安全边界，因为云端本身就在消费者单位组织的网络安全边界之内，但是人们更倾向于给托管私有云的云端再建立一道安全防护边界。

3．安全职责共担

由于云服务提供商和云服务消费者对于云端的计算资源具有不同的控制范围（如图 3-11 所示），与传统 IT 系统完全由一家单位组织控制不同，云计算系统要求云服务提供商和云服务消费者在设计、建设、部署和操作云系统时必须通力合作，双方共同努力才能构建一个安全的云计算环境。我们必须仔细分析每一项安全控制措施，再结合具体的云服务模式，确认其最佳的责任方（云服务提供者或云服务消费者）。比如对于 IaaS 云服务模式来说，具备初始系统权限的用户账户管理通常由 IaaS 云服务提供商负责，而部署在 IaaS 环境（如虚拟机）中的应用程序的账户管理则由云服务消费者负责。

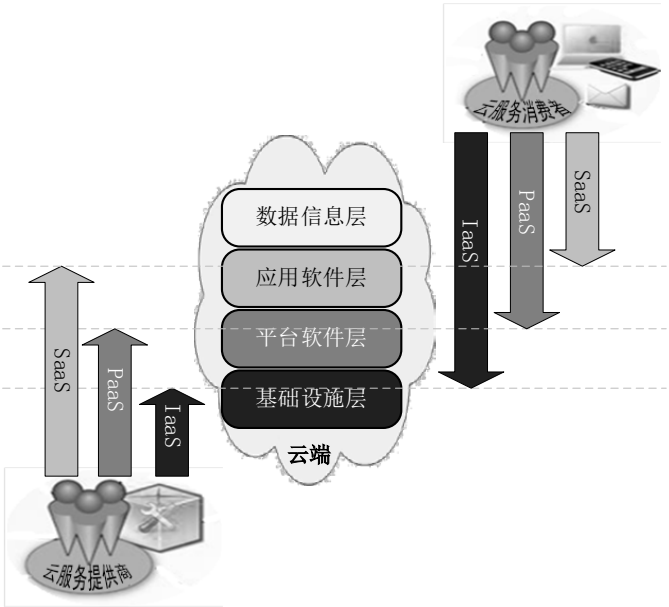


图 3-11　云服务提供商和云服务消费者的控制范围

3.3.4　隐私保护

3.3.3 节的“安全保障”涵盖了隐私保护的内容，但是在云计算时代，个人信息、操作行为和习惯都存储在云端或者在云端留下印迹，所以隐私保护的重要性是前所未有的，有必要单独拿出来讨论。

隐私是指关于个人或者单位组织本身的信息，信息的主体不愿意公开而且法律没有规定必须公开的信息。隐私包括个人信息（PI）和个人身份信息（PII），PII 描述了个体的静态信息，如姓名、年龄、性别、住址、电话等，而 PI 描述了与个体关联的周边信息，如家庭、存款、社会关系、行动路线、驾照号码等。主体的身份不同，隐私的具体内容也不尽相同。比如，普通民众的个人财产是隐私，但是公务员的财产就不是隐私（法律规定必须公开）；再比如，企业的财务报表是隐私，但是上市企业的财务报表就不是隐私。

如果云服务提供商收集了消费者的身份信息或者操作行为和习惯模式，那么它们必须确保这些隐私信息能得到可靠保护，除消费者本人外，其他任

何人或机构都不能获取这些信息。对于全球性的或者跨国经营的云服务提供商来说，隐私保护的问题变得异常复杂，因为各国关于隐私保护的法律都不尽相同。

3.4 云计算审计员

云计算审计员能对云计算利益相关者开展独立检查并发布评估结果，审计的核心任务就是通过对客观证据的审查来评估是否符合预设的标准。针对云服务提供商的审计主要包括安全审计、隐私保护审计和性能审计等。

1．安全审计

云计算审计员评估云服务提供商是否具备足够的且准备妥当的安全控制措施，以及是否严格遵守切实可行的安全流程。例如，云计算审计员审查云服务提供商是否遵守了 ISO 27001 安全标准。

2．隐私保护审计

隐私保护审计主要检查云服务提供商是否保护了个人信息（PI）和个人身份信息（PII）。

3．性能审计

云服务提供商必须满足在服务水平协议（SLA）中列举的服务质量(QoS）的要求，但是云服务消费者往往抱怨其利益受到损害，而云服务提供商也常常表示无辜，只有通过第三方独立开展性能审计才能消除供/需双方的分歧。

3.5 云服务代理商

随着云计算的不断发展，云服务公司提供的产品越来越多，如何整合这

些产品以满足用户的需求也变得越来越复杂。例如，登录亚马逊官方网站，你会发现有上百种云服务产品，有的产品可以单独租赁，但有的需要整合其他产品才能发挥最佳效果。所以，不少潜在云服务消费者宁愿把需求告诉给云服务代理商，由云服务代理商全权负责选购产品并做最佳的整合，也不愿耗费时间和精力直接跟众多的云服务提供商打交道。云服务代理商管理云服务的使用、性能和交互，并且协调云服务供/需双方的关系。随着云服务市场的繁荣，可以预计，将来还会出现云服务一级代理、二级代理，甚至更多级代理的现象。

通常，云服务代理商的主要工作有以下三个方面。

（1）服务中介：云服务代理商处于云服务提供商和云服务消费者之间，他是云服务提供商的客户，又充当云服务消费者的云服务提供商。云服务代理商尽可能地改善一些云服务产品并给消费者提供增值服务，这些改善措施包括但不限于强化安全、完善性能报告、优化身份管理、做好本地化（语言、本地人的习惯、本地法律法规等），以及增强用户的交互体验等。服务中介如图 3-12 所示。

图 3-12　服务中介

（2）服务聚合：云服务代理商把多个服务整合成一个或几个新的服务，以满足用户复杂的需求，同时提供数据集成，使得云服务消费者和多个云服务提供商之间的数据移动是安全的。服务聚合如图 3-13 所示。

（3）服务套利：“套利”意为快速买卖以赚取差价。这里的服务套利类似于服务聚合，但是被聚合的服务是不固定的，云服务代理商可随时在多个云服务提供商的产品中筛选最好的服务。比如，云服务代理商可以使用信用评分措施来衡量和挑选分数最高的云服务提供商。

图 3-13　服务聚合

3.6　云服务承运商

云服务承运商负责在云服务提供商和云服务消费者之间建立连接媒介，以便把云计算服务产品从云服务提供商那里转移到云服务消费者手中。云服务的分发工作通常由网络和电信运营商或者专门的传输代理机构完成。传输代理机构是指提供诸如大容量磁盘这种物理的传输存储媒介的业务机构。为了保证服务质量，云服务提供商会和云服务承运商签订服务水平协议（SLA），还可能要求云服务承运商提供专线，以保证云服务供需双方的连接安全。

第4章

“云”组件

目前，最主要的云服务产品是 IaaS 虚拟机，一套完整的对外出租虚拟机的 IaaS 云计算解决方案必须解决下面这个问题：

如何运行和管理大量的虚拟机并让远方的用户自助使用这些虚拟机？

问题中的三个动词“管理”、“运行”和“使用”意味着一个 IaaS 云计算系统包含以下三部分，如图 4-1 所示。

（1）**虚拟化平台（硬件、虚拟软件）**——解决如何运行虚拟机的问题。

（2）**管理工具**——解决如何管理大量虚拟机的问题，包括创建、启动、停止、备份、迁移虚拟机，以及计算资源的管理和分配。

（3）**交付部分**——解决如何让远端的用户使用虚拟机的问题。

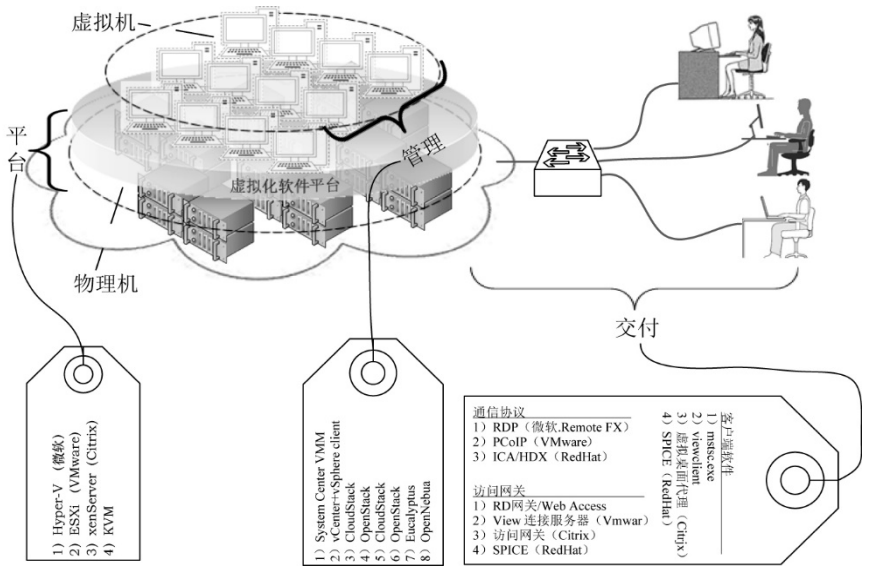


图 4-1　IaaS 云计算解决方案组成部分

1．虚拟化平台

虚拟化平台中的硬件部分主要指服务器、存储和网络。对于服务器，大的云计算提供商倾向于自己定制，这种针对特定应用定制的服务器具备更高的计算效率和更低的成本，因此目前的通用服务器硬件厂商面临很大的市场压力。有一些云计算厂商推出云计算一体机，即把平台和管理两部分打包成一台服务器出售，用户购买这样的一体机后，可以直接向员工交付计算机桌面。

平台中的虚拟软件安装在物理机或者操作系统上，然后通过它创建若干个虚拟机并运行这些虚拟机。当然，虚拟机中还要安装操作系统，如 Windows 8、Linux 等。一个云端可能有很多台服务器，每台服务器上又有很多个虚拟机，那么如何管理这些虚拟机呢？这就是云管理平台的任务了。

2．管理工具

管理工具就是一套软件，是用来管理云端的资源（服务器、存储、网络）和虚拟机的。虚拟机是资源申请的基本单位，因此管理平台的核心任务就是管理虚拟机，即进行创建、销毁、启动、关闭、资源分配、迁移、备份、克隆、快照及安全控制等操作。这一点非常类似于传统操作系统的进程管理，所以有人说云管理工具就是云操作系统，即用来管理云端资源（计算、存储、网络），虚拟机是云端资源分配的主体，而传统的操作系统是用来管理计算机资源（CPU、内存、硬盘等）的，进程是计算机资源分配的主体。但是把云管理工具称作云操作系统不太恰当，称其为平台也勉勉强强，所以还是称其为虚拟机管理工具比较合适。

3．交付部分

云计算的本质是计算与输入/输出分离，那么处于远方的云端资源如何交付给用户呢？换句话说就是，用户如何使用云端的计算资源（如计算机桌面），这是很关键的问题。

交付主要由三部分组成：一是通信协议，二是访问网关，三是客户端。通信协议就是规定终端与云端的通信规则，比如中国香港警匪片的警察用对讲机通话时，以“Over”作为本人说话的结束语，对方听到“Over”后开始说话，这是一种最简单的通信协议。协议的好坏与终端用户的体验息息相关，也是最具技术含量的部分，目前也就三四家大企业才有能拿得出手的协议。

访问网关相当于云端的大门，终端用户必须由此“门”进入云端。客户端是指安装在云终端上的软件，专门负责与云端的通信——接收用户的输入并发到云端，然后接收云端的返回结果并显示在云终端屏幕上。一台云终端

上可以安装多个不同公司发布的客户端，不同的客户端，一般通信协议也是不同的，这样的云终端具备接入多个由不同提供商运营的云端的能力。比如笔者开发的云终端，既可以接入微软的 RDP 协议云端，又可以接入 VMware 的 PCoIP 协议云端，还可以接入 Citrix 的 HDX 协议云端，以及红帽的 SPICE 协议云端。

上面提到的是云计算系统的三个“主心骨”，实现了云计算的基本功能，但是在生产环境中，还必须满足性能和产能的要求，因此下面这些附加部分也是不可或缺的：负载均衡、故障转移、身份认证、权限控制、入侵检测等。

无论如何，云端本身还是一个 IT 系统，仍然遵循九层逻辑架构，每一层都由若干组件构成。下面对构成云端的各个组件做一个简单介绍，更详细的资料请浏览相应的官方网站。

4.1 开源组件

4.1.1 开源计算项目

开源软件，就是软件源代码公开，任何人都可以下载和免费使用并遵循某种开源协议的软件。而开源计算项目（Open Compute Project，OCP）是一套开源硬件，它是由 Facebook 公司主导，众多 IT 巨头公司参与的针对数据中心定制的硬件设计规范，包括机房、机柜、服务器、存储、网络设备的定制设计规范，以及云端硬件的管理规范。只要一个人有资金并且有时间，参考这些规范和购买相应的配件，也能单独做出非常可靠的东西（如设计合理的机房、服务器、交换机等）。

开源计算项目针对的是 IT 架构的第一层——基础设施层，目前大型的云服务提供商或者社交网站热衷于定制自己数据中心的硬件，包括机柜、服务器、存储和网络设备。由于这种硬件是针对应用软件的特点定制的，所以计算效率非常高。完成相同的计算任务，相比其他硬件，能源消耗要低很多。

云端硬件发展的另一个趋势是云设备提供商提供的云计算一体机（具体参见前面的章节）。

开源计算项目中的主板和服务器设计规范包括针对 ARM、AMD、Intel 的 CPU 的主板的设计规范，高可用性服务器、片上系统和电源的设计规范，同时提供了物理结构的 CAD 设计图纸。Penguin Computing 公司生产的主板和服务器如图 4-2 所示。

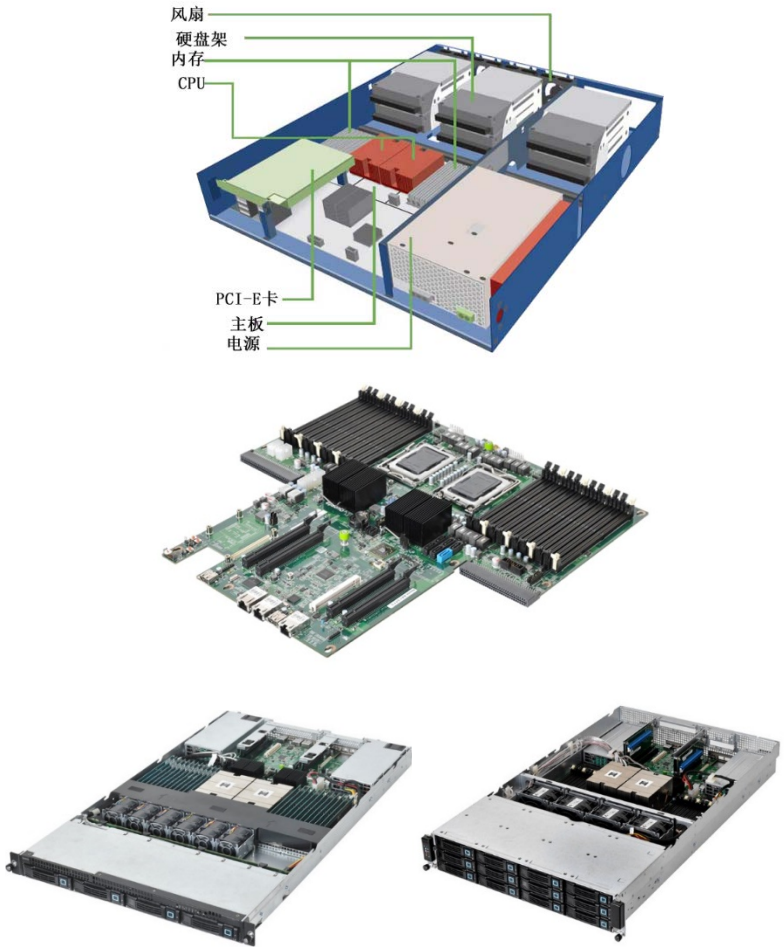


图 4-2　Penguin Computing 公司生产的主板和服务器

开源计算项目中的子项目“存储”，分别就冷存储、Fusion-io 等发布设计

规范，图 4-3 所示的存储器就是参考此规范设计并生产的。

图 4-3　存储器

另外，也有一些开源存储软件的解决方案，比较有名的有 OpenIndiana、NexentaStor、FreeNAS、OpenFiler，性能也是依次降序排列的。当然，如果只是自己使用，那么建议采用 Oracle Solaris 11+Napp-It 的最佳方案。这些解决方案的核心都采用了 ZFS 文件系统。ZFS 文件系统最早是由 Sun 公司开发的，Oracle 公司收购 Sun 后，从 ZFS 2.8 版本之后就不再开源了，闭源的 ZFS 增加了不少新功能，如加密，但是有人在 ZFS 2.8 版本的基础上继续开发新功能，这样开源的代码就与正宗的 ZFS 代码分道扬镳了。

除这些专门的存储设备外，在云计算和大数据时代，分布式存储系统具备更大的优势，采用成百上千台甚至上万台计算机组成海量数据存储系统，拥有更高的带宽、更高的 IOPS 和更好的可用性，容量也具备更佳的伸缩性。例如 Ceph、Lustre、GlusterFS、pNFS、GPFS，都是一些优秀的企业级开源分布式存储项目。

4.1.2　虚拟化

在云端，虚拟化属于基础设施层，具体包括服务器虚拟化、网络虚拟化

和存储虚拟化，虚拟的目的是为了池化物理资源以便用户能随时获取并使用。在服务器虚拟化领域，VMware（早被 EMC 公司收购，后来戴尔公司又收购了 EMC 公司）已耕耘了很多年，把持了大部分的市场。后来者微软、思杰都想有所突破。微软推出 Hyper-V，并在短时间内进行了版本快速更替，其与操作系统紧密捆绑、简单易用的特征，使得采用微软的云计算解决方案成为最简捷的商业化方案。另外，微软拥有从操作系统到应用软件的全部产品线，再加上成千上万个基于 Windows 的第三方应用软件，微软具备很强的竞争优势。

思杰自从收购 XEN 并进军虚拟化市场以来，表现平平，于是被逼开源它的 XenServer 产品，使得 XEN 重新回到开源社区。另外，思杰还把云管理工具 CloudPlatform 卖掉，重回 OpenStack 社区。在开源社区，美国的红帽公司也在一直尝试整合开源资源并形成自己的一整套云计算产品，其核心是已经并入 Linux 内核的 KVM 虚拟机。开源社区中的另一个不容忽视的成员就是 Oracle VM。红帽与微软类似，拥有从操作系统到应用软件的完整产品线，而且全部是开源的。云计算提供商都是技术性的公司，它们更倾向于采用开源的云计算解决方案，稳定、可靠、低成本注定了红帽公司在未来的云计算市场会有更大的作为。下面对开源的服务器虚拟化组件做简单介绍。

1．XenServer

XenServer 是基于 Xen 的虚拟平台，直接安装在物理的计算节点上，然后在上面创建更多的虚拟机。XenCenter 是云计算管理平台，通过它来管理众多的安装了 XenServer 的服务器（计算节点）、网络、存储及虚拟机。

XenCenter 通过计算节点池来管理服务器，属于同一个池的服务器共享存储，而且服务器最好是相同配置的机器，池里的虚拟机保存在共享存储中，一个虚拟机允许在池中的任何一台服务器上运行。

XenCenter 服务器管理如图 4-4 所示，包含三个服务器池，池中的每台服务器必须安装 XenServer，这些池通过一台安装了 XenCenter 的服务器来集中管

理：创建池、把服务器添加到池或者从池中移除、在池中创建虚拟机、创建虚拟机模板、热迁移虚拟机、创建虚拟机快照等。采用 XenServer 搭建最简单的云端就是只有一台安装了 XenServer 的服务器，然后在一台个人计算机上安装 XenCenter 来管理这台 XenServer 服务器。

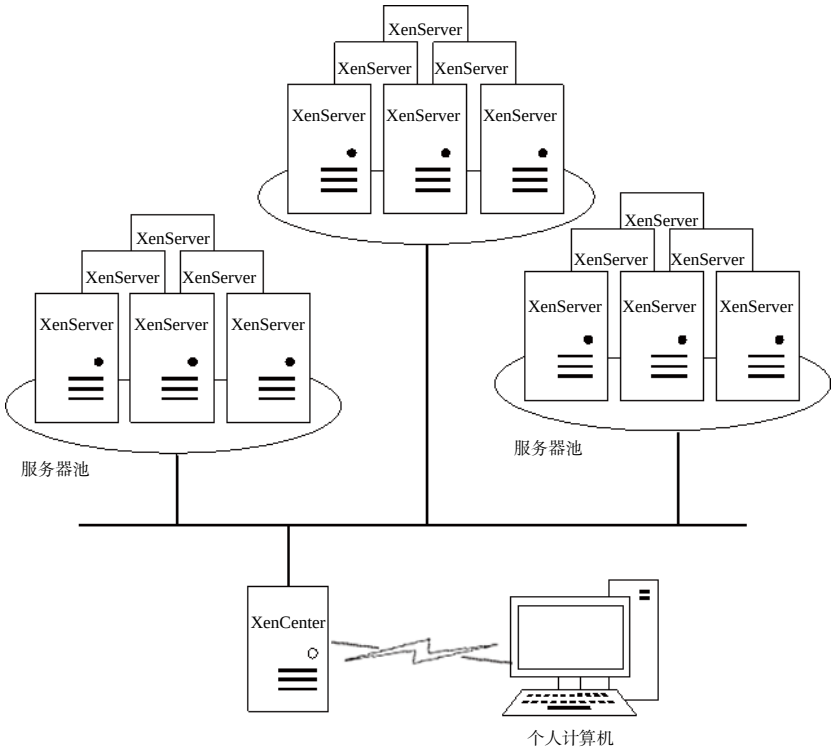


图 4-4　XenCenter 服务器管理

2．Red Hat Virtualization（RHV）

这是红帽公司发布的完整的虚拟化解决方案，它集成了 RHEL 操作系统、KVM 虚拟化软件和 oVirt 虚拟化管理工具。红帽公司并以此为基础推出了私有云套件 Red Hat Cloud Infrastructure、共/私有云和虚拟化环境的统一管理平台 Red Hat CloudForms，以及混合云方案套件 Red Hat Cloud Suite。RHV 包含虚拟化管理工具（Red Hat Virtualization Manager，RHVM）、虚拟化软件（Red Hat Virtualization Host，RHVH）和虚拟化存储（Red Hat

Virtualization Storage，RHVS）三个主要组件。其中，RHVH 直接安装在物理计算节点上，然后可在 RHVH 上运行虚拟机，RHVH 架构如图 4-5 所示，通过 RHVM 集中管理众多的 RHVH 服务器。逻辑上，先有数据中心（DataCenter），数据中心包含多个集群（Cluster）、存储池和网络。一个集群由多台安装了 RHVH 的服务器组成，虚拟机创建在集群上，一台虚拟机允许运行在集群中的任何一台服务器上。存储是基于数据中心层面的，“数据中心”是一个逻辑概念，不一定是局限在地理位置上的机房，也可能跨机房，或者在一个机房里存在多个数据中心。

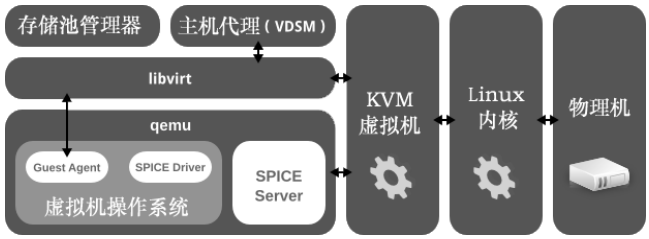


图 4-5　RHVH 架构

用户通过 SPICE 协议连接到虚拟机（需要安装 SPICE 客户端软件），访问格式是 SPICE://<计算节点 IP>:端口，这里的 IP 是虚拟机的宿主服务器的 IP 地址，每台虚拟机在宿主机上对应一个端口，具体内容参考 5.3 节的 SPICE 协议。虽然红帽声称 RHV 是开源的，但是仍需要购买订阅服务才能下载，对于不想花钱又想使用虚拟化的用户来说，可以采用完全开源的替代产品 oVirt。

3．Oracle VM Server

Oracle VM Server 是由 Oracle 公司发行的、集成 Linux 内核和 Xen 的服务器虚拟软件，因此与 XenServer 大同小异，不过它不但支持 X86 机器，还支持 SPARC 机器。配套发行的 Oracle VM Manager 云管理工具采用 Java 语言开发，不过其视野中的云端架构相对而言比较复杂。Oracle 公司的产品线横跨硬件和软件，所以它推出的云计算方案比较全面，从 IT 系统的基础设施到应用层都有相应的配套产品。

4．OpenVZ

OpenVZ 容器是一个轻量级虚拟机，容器里是一个完整的操作系统环境，它的主要用途是充当虚拟私有服务器，即 VPS。OpenVZ 是在 Linux 操作系统上（修改了宿主机上的 Linux 内核）创建隔离的运行环境，每个运行环境拥有自己的 IP、Root 用户、根目录和运行库，但共享操作系统内核，即容器中不能再安装操作系统，允许安装更多的应用软件，OpenVZ 架构如图 4-6 所示。现在很多的 VPS 租赁公司几乎都是采用它来创建 Linux 的 VPS。Windows 的 VPS 目前有 Windows Server Container 和 Hyper-V Container。相对于其他虚拟机软件而言，在相同配置的物理机上，用 OpenVZ 可以运行更多的容器，从而大大降低每个用户的成本。在公有云中，更多的公司倾向于采用容器。类似的开源容器还有 Linux-VServer、LXC、FreeBSD jail、Solaris Containers。

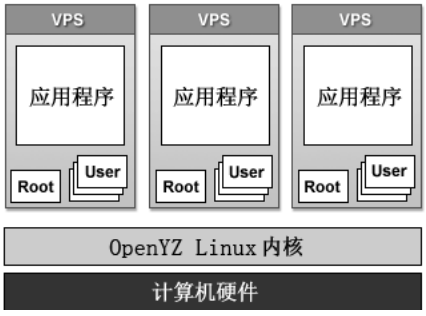


图 4-6　OpenVZ 架构

但是 2013 年兴起的 Docker 容器更轻便（容器只运行一个应用程序），其在谷歌公司的大力推动下发展迅速，而且谷歌发布了构建于 Docker 之上的开源的 Kubernetes 管理平台，使得管理运行在成千上万台计算机上的数十万个 Docker 容器变得异常轻松和简单。容器和虚拟机的区别如图 4-7 所示。

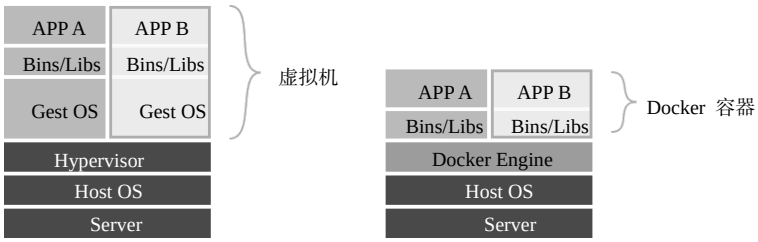


图 4-7　容器和虚拟机的区别

5．VirtualBox

这是 Oracle 发布的虚拟机软件，单机版，是 VMware Workstation 的替代品，能安装在 Windows、Linux、Mac OS X 和 Solaris 操作系统上。在由 VirtualBox 创建的虚拟机上，可以安装 Windows、Linux、FreeBSD 操作系统。VirtualBox 支持虚拟机快照、睡眠和分组、虚拟机导入/导出、虚拟机克隆等。

6．oVirt

为红帽 RHV 的社区版，是一套完整的开源虚拟化管理工具，集成 KVM 虚拟化软件和红帽 RHVM 的 Web 管理界面，再结合 GlusterFS 分布式存储工具，能实现几十台服务器的虚拟化应用环境，能满足拥有近千名员工的企事业单位的虚拟化需求。oVirt 架构如图 4-8 所示。

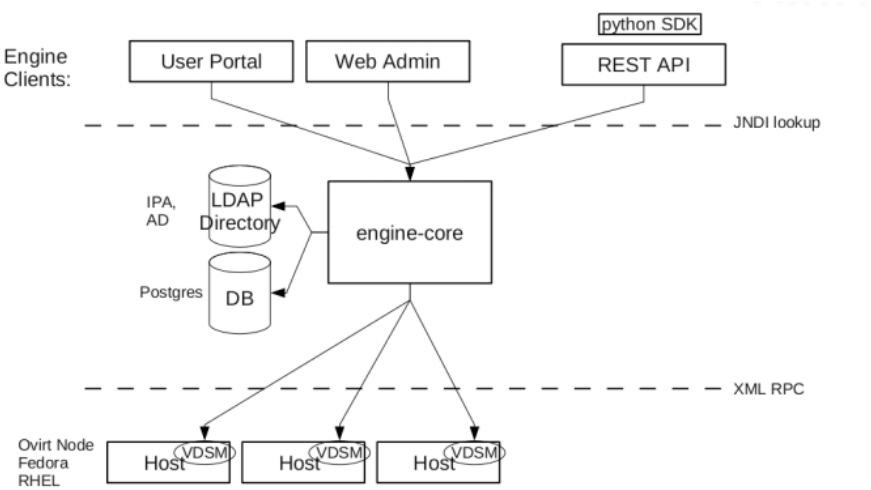


图 4-8　oVirt 架构

4.1.3 操作系统

1．Linux 内核

Linux 内核诞生于 1991 年，目前已超过 2000 万行代码，价值达数百亿美元。以它为核心推出的操作系统发行版最为流行的有 RedHat、SUSE、Ubuntu、Android、CentOS、Debian 等，大概有上百种。如果不特别指定，这些发行版统称为 Linux 操作系统。Linux 操作系统开源免费、稳定可靠、病毒少、性能高，作为平台软件，被广泛应用在云端、数据中心。目前除微软外，全部云中心都采用 Linux 操作系统。CentOS 是 RedHat 的克隆版，应用比较广泛，网上具有丰富的资料，现已被红帽收购，成为 RHEL 的实验版。

2．FreeBSD

FreeBSD 是由美国加州大学伯克利分校开发的免费开源的操作系统，号称世界上最稳定、最高效、最安全、版权限制最少的操作系统。其他公司或者个人可以下载源代码，进行修改后不开源。比如一些网络设备生产商修改源代码后集成到自己的产品中，修改后的代码不开源是不违法的，这点与 Linux 操作系统不同。FreeBSD 是最理想的网络服务器操作系统之一。鼎鼎有名的开源 NAS 项目 FreeNAS 就采用了这个操作系统。

3．Illumos 内核

Oracle 收购 Sun 公司之后，关闭了 OpenSolaris 开源项目，推出闭源的 Solaris 11 操作系统和一个简化版的 Solaris 11 Express。Solaris 11 可以从官方网站下载，如果商用，就需要购买服务，个人和实验室可以免费使用。OpenSolaris 的原班人马几乎都离开了 Oracle 公司，分别加入了以 Illumos 为核心的各个厂商，一时间发布了许多个开源版本，如 Napp-it to GO、NexentaStor、OmniOS、SmartOS、StormOS、Illumian、OpenIndiana，可分别应用于后台服务器、桌面计算机、存储、嵌入式等领域。这些发行版都以 Illumos 为核心，这一点类似于各种 Linux 发行版，如红帽、乌班图、Debian 等 Linux 操作系统发行版都是以 Linux 为核心的。利用 Napp-it to GO 自己搭建 NAS 存储是一

个低成本的存储解决方案，其中核心是 ZFS 文件系统。但是开源的各个操作系统版本中 ZFS 文件系统只到 2.8 版本，更高的版本被 Oracle 公司闭源了，Oracle 公司现在已经推出了 ZFS 3.4 版本。如果自己使用，则可以采用 Solaris 11 + Napp-it 组建企业级的 NAS 存储系统。不过 Napp-it 的界面太过复杂，用户体验非常糟糕。

4.1.4 数据库

从早期的 SQL 关系数据库开始，陆续诞生了 NoSQL 和 NewSQL 两大类数据库。SQL 数据库具备良好的 ACID 特性，同时兼容 SQL 标准，但是不易扩展；NoSQL 具备很好的扩展性但是缺少实时一致性和 SQL 标准；而 NewSQL 算是二者的折中，是后起之秀，在大数据领域应用广泛。

表 4-1 列出了每一类数据库的前 3～5 名，排名会不断变化，实时排名结果请参见 http://db-engines.com/en/ranking 网站。

表 4-1 各类数据库排名

大 类	类 别	前 3～5 名	说 明
SQL	关系数据库	Oracle、MySQL/MariaDB、SQL Server、PostgreSQL、DB2	遵循“表—记录”模型。按行存储在文件中（先第 1 行，然后第 2 行……）
NoSQL	时序数据库	InfluxDB、Kdb+、Graphite、RRDtool、Prometheus	存储时间序列数据，每条记录都带有时间戳，如存储从感应器采集到的数据
	键/值数据库	Redis、Amazon DynamoDB、Memcached、Hazelcast、Ehcache	遵循“键—值”模型，是最简单的数据库管理系统
	文档数据库	MongoDB、Amazon DynamoDB、Couchbase、CouchDB、MarkLogic	无固定结构，不同的记录允许有不同的列数和列类型。列允许包含多值，记录允许嵌套
	图数据库	Neo4j、Datastax Enterprise、OrientDB、ArangoDB、Virtuoso	以“点—边”组成的网络（图结构）来存储数据

续表

大　类	类　别	前 3～5 名	说　　明
NoSQL	搜索引擎	Elasticsearch、Splunk、Solr、MarkLogic、Sphinx	存储的目的是为了搜索，主要功能是搜索
	对象数据库	Caché、Versant Object Database、db4o、ObjectStore、Matisse	受面向对象编程语言的启发，把数据定义成对象并存储在数据库中，包括对象之间的关系，如继承
	宽列数据库	Cassandra、HBase、Datastax Enterprise、Accumulo	按照列（由“键—值”对组成的列表）在数据文件中记录数据，以获得更好的请求及遍历效率。一行中的列数允许动态变化，且列的数目可达数百万，每条记录的关键码也不同，支持多值列
NewSQL		CockroachDB、Altibase、NuoDB、VoltDB	SQL 和 NoSQL 的折中，同时具备良好的 ACID，兼容 SQL 标准，容易大规模扩展

下面对排名靠前的开源数据库做一些简单介绍。

1．MySQL/MariaDB

2008 年美国 Sun 公司花费 10 亿美元收购 MySQL，一年后 Oracle 公司又花费 60 亿美元收购了 Sun 公司，从此 Sun 公司的服务器、操作系统、MySQL 等产品线全部归属 Oracle 公司。拉里·埃里森奉行的经营哲学是“竞争不过它，我就买了它”。竞争不过，说明竞争对手的产品更具优势，所以收购它，从而变成自己的优势。几十年来，Oracle 从一家小型数据库公司变成今天覆盖硬件、平台软件、数据库、中间件、应用软件各个层次产品线的 IT 巨无霸，靠的就是美国的良好环境和总裁奉行的收购策略。

MySQL 数据库占据中小型数据库应用市场的半壁江山，在这块市场，Oracle 数据库明显占下风，巅峰时世界上超过 70%的网站后台都采用 MySQL 数据库。但是自从被 Oracle 公司收购后，MySQL 发展明显趋缓，是继续开源还是闭源，Oracle 公司一直没下定论。于是 MySQL 的原班人马陆续离开 Oracle

公司，另立炉灶，推出了 MariaDB 开源数据库。MariaDB 继承了 MySQL 小巧精悍、简洁高效、稳定可靠的特征，并与 MySQL 保持兼容。时至今日，已有谷歌、Facebook 等知名企业把应用从 MySQL 切换到了 MariaDB 上，各种 Linux 发行版的操作系统默认数据库都开始采用 MariaDB；而苹果公司反应更快，当 Oracle 公司收购 Sun 公司时，就切换到了 PostgreSQL 数据库。截至 2014 年年末的数据库综合排名，MySQL 虽然位居第二名，但是其表现出来的颓势较明显，而 MariaDB 却具备强劲的生命力。

MariaDB 是一个开源的免费的关系数据库，截至发稿前的最新版本是 10.0.15，安装包可从 https://downloads.mariadb.org 网站下载。整个安装包大约 200MB，几乎能在所有的操作系统上安装和运行，与 Oracle 数据库、SQL Server、DB2 等商业数据库相比，算是短小精悍了。对于非数据库型的 SaaS 云提供商来说，云端采用 MariaDB 是最好的选择，可以轻松搭建数据库集群。

2．PostgreSQL

PostgreSQL 能被苹果公司大量使用并替换早期的 MySQL 数据库，说明其一定有不俗的表现。在 2014 年年末的数据库排行榜上，PostgreSQL 位居第四位，号称世界上最先进的、最安全的开源数据库，也是面向对象的关系数据库。华为内部局域网接入控制系统 SPES 也采用 PostgreSQL 数据库，在 Linux 操作系统上运行，极少出问题。经过十多年的磨砺，如今 PostgreSQL 数据库具备如下特点。

面向对象，安全性极高，能运行在全部主流的操作系统上，完全 ACID 兼容，支持外键、连接、视图、触发器和存储过程（多语言），支持最新的 SQL 2008 标准的数据类型，包括整型、数字型、字符型、布尔型、可变字符型、日期型、时间戳、自增长型，以及各种二进制对象（如图片、声音和视频等），提供了各种流行语言的编程接口，如 C/C++、Java、.NET、Perl、Python、Ruby、Tcl、ODBC 等。

PostgreSQL 数据库同时具备如下众多令人振奋的优点。

在线备份、数据域、表分区、表空间、嵌套事务、时刻点恢复、两段式

提交、多版本并发控制、子查询等。

在 PostgreSQL 中，数据库大小没有限制，表大小上限为 32TB，一条记录的大小上限为 1.6TB，字段大小上限是 1GB，一个表包含的记录数没有限制，一条记录的字段数上限为 1600 个，一个表上创建的索引数目没有限制。

另外，SQLite 是一个非常小型的 ACID 兼容的关系数据库，适合嵌入式系统。

3．InfluxDB

InfluxDB 是一个开源的时间序列数据库，能应付极高的写和查询并发数，主要用于存储大规模的时间戳数据（每条记录自动附加时间戳），如 DevOps 监控数据、应用系统运行指标数据、物联网感应器采集的数据及实时分析的结果数据等。InfluxDB 具备如下特征：

- 全部用 Go 语言编写，并被编译成单一运行程序，无须第三方依赖。
- 简洁、高效地写和查询 HTTP(S)编程接口（API）。
- 通过插件能与其他的数据采集工具集成，如 Graphite、collectd、OpenTSDB。
- 可以搭建高可用性的 InfluxDB 环境。
- 量身定制化的类 SQL 语言。
- 允许给序列数据附加标签来创建索引，以便快速高效查询。
- 通过定义策略轻松实现自动失效过时的数据。
- 基于 Web 的管理界面。

对于一个具体的时间序列应用来说，除存储外，还需要集成数据采集、可视化和告警功能。为此，InfluxData 社区相应提供了 Telegraf（数据采集）、Chronograf（数据可视化）、Kapacitor（告警）三个开源项目，再加上 InfluxDB，能部署成一个完整的时间序列应用系统（简称为 TICK）。这四者的关系如图 4-9 所示。

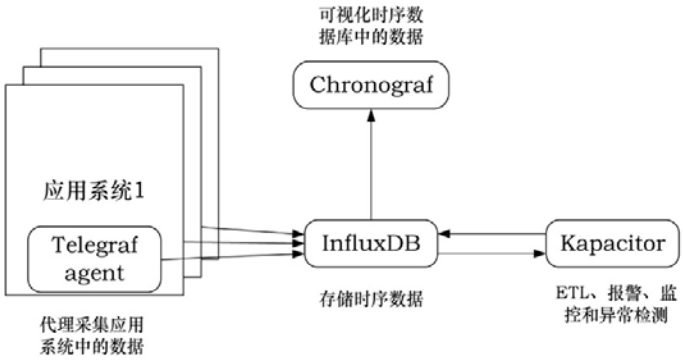


图 4-9　时间序列应用系统组成结构

4．Redis

Redis 是遵循 BSD 开源协议的存储系统，数据存储在内存中，因此具备极高的性能，可用作数据库、缓存和消息中间件。Redis 支持多种类型的数据结构，如字符串、散列、列表、集合、带范围查询的有序集合、位图、HyperLogLogs 和带半径查询的地理空间索引。Redis 内置了复制、脚本语言编程、最近最少使用（LRU）淘汰、事务，以及不同级别的磁盘持久化等功能，通过 Redis Sentinel 和集群自动分区机制实现高可用性。Redis 采用 C 语言编写，能运行在 Windows、Mac OS X、Linux、Solaris 等操作系统上，不过 Linux 是其最佳的运行平台，无须第三方依赖，它提供了最广泛的编程语言接口。

5．MongoDB

MongoDB 是排名第一的文档数据库，属于 NoSQL 大类，诞生于 2009 年，正好是云计算兴起的前夜。MongoDB 采用 C++语言开发，能运行在 Windows、Mac OS X、Linux、Solaris 操作系统上，提供了绝大部分计算机语言的编程接口。保存在 MongoDB 中的一条记录称为一个文档，类似 JSON 语法，例如：

```
{
   name: "张三",
   age: 26,
   address: "科发路1号",
   groups: ["新闻","体育"]
```

}

从上面的例子可以看出，一个文档就是“键：值”对的集合。MongoDB的主要优势包括：高性能、富查询语言（支持 CRUD、数据聚合、文本搜索和地理空间查询）、高可靠性、自动伸缩架构、支持多存储引擎。MongoDB适合文档存储、检索和加工的应用场合，如大数据分析。

6．Neo4j

Neo4j 是一个用 Java 语言开发的图数据库，它将结构化数据存储在由“点—边”组成的网络（数学术语叫“图”）上而不是表中。与关系数据库相比，图数据库擅长处理大量复杂、互连接、低结构化的数据。这些数据变化迅速，需要频繁查询，如果使用关系数据库，那么这些查询会导致大量的表连接，因此会产生性能上的问题。Neo4j 重点解决了拥有大量连接的传统关系数据库在查询时出现的性能衰退问题。Neo4j 还提供了非常快的图算法、推荐系统和 OLAP 风格的分析，而这一切在目前的关系数据库系统中都无法实现。Neo4j 提供了广泛使用的 REST 接口，能够方便地集成到基于 Java、PHP、.NET 和 JavaScript 的环境里。

Neo4j 使用 Cypher 语言进行增删改操作，例如：

```
CREATE (e1:员工 { 姓名: "张勇", 国籍: "中国" , 年龄: 30 })
CREATE (e2:员工 { 姓名: "John", 国籍: "美国" , 年龄: 35 })
```

创建了两个节点：变量“e1”和“e2”，标签“员工”，属性“{……}”。

```
CREATE (e1)-[:领导]->(e2)
```

创建了一条边，如图 4-10 所示。

图 4-10　创建一条边

7．Elasticsearch

Elasticsearch 是一个高度可扩展的开源全文搜索和分析引擎，用于存储、搜索和近乎实时地分析大规模数据。Elasticsearch 常被人们用于构建具备复杂搜索功能的应用系统，比如维基百科和优步的搜索系统，再比如网店中搜索商品、商业智能、日志收集与分析、价格告警等。利用 Elasticsearch 实现的数据采集、存储和可视化套件 ELK（Elasticsearch、Logstash、Kibana）在日志分析领域应用最为广泛。

8．Caché

大概 20 年前，人们提出了后关系型数据库的概念，目前已有不少产品，其中由 InterSystems 公司开发的 Caché 是后关系型数据库中的领头羊，它提供 3 种方式访问数据：对象访问、SQL 访问、直接对多维数据数组访问，而且 3 种访问方式能够并发访问同一数据。多维数据结构和多种数据库访问方式构成了后关系型数据库的基础。Caché 数据库具备如下一些特点：

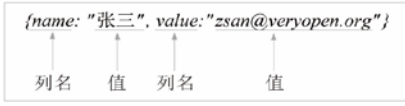
- 速度快。Caché 数据库在同等条件下查询相同数据，比 Oracle 等普通数据库要快 7 ~ 20 倍。
- 节省存储空间。存储空间可节省 2/3，而基于 Caché 的应用程序，比原来基于关系型数据库的应用程序运行有可能快 20 倍。
- 使用简单。Caché 数据库支持标准 SQL 语句，因此不太熟悉 M 语言的用户依然可以轻松对数据库中的数据进行操作。
- 接口容易。Caché 数据库支持 ODBC 标准接口，因此在与其他系统进行数据交换时非常容易，同时 Caché 亦可以将数据输出为文本文件格式，以供其他系统访问调用。
- 真正的三层结构。Caché 数据库能够在真正意义上实现 3 层结构，实现真正的分布式服务，便于做横向扩张。
- 对象型编辑。Caché 数据库是真正的对象型数据库，开发时用户可直接用数据库定义自己想要的对象，然后再在其他开发工具中调用该对象的方法和属性即可完成开发工作，非常方便。

- 支持 Web 开发。Caché 数据库提供自带的 Web 开发工具，使用、维护非常方便，符合当今软件业发展的趋势。
- 价格便宜。Caché 数据库的价格比 Oracle 数据库要便宜很多。

Caché 数据库在欧美的医疗系统使用最为普遍，被医疗界公认为首选的数据库，国内中山大学第三附属医院、深圳南山医院、北京安贞医院等也采用 Caché 数据库。

9．Cassandra

Cassandra 是在谷歌的 BigTable 基础上发展起来的 NoSQL 数据库，由 Facebook 于 2008 年用 Java 语言开发，目前被贡献给 Apache 基金会。Cassandra 被称为“列数据库”，这里的“列”不是指关系数据库中一个表中的列，而是由“键—值”对组成的列表（语法与 Python 语言中的列表相同），如：



Cassandra 中一行数据语法是“一行的键={列,列,…}”，一行可包含上百万列，比如：

Cassandra 中的列族（ColumnFamily）格式是“列族名={若干行}”，一个列族可包含几十亿行，例如：

```
UserProfile ={
    wlm={ emailAddress :"casandra @veryopen . org" ,  age: " 20" }
    TerryCho= { emailAddress :" terry . cho@ veryopen . org " ,  gender :" 男" }
    John = { emailAddress :"cath@veryopen . org" , age: " 20" , gender :" 女" , address :" 中山" }
}
```

一个 Cassandra 运行实例管理很多键空间（Keyspace），Keyspace 相当于关系数据库管理系统中的数据库，一个键空间包含很多列族。键空间、列族、行、列的关系如图 4-11 所示。

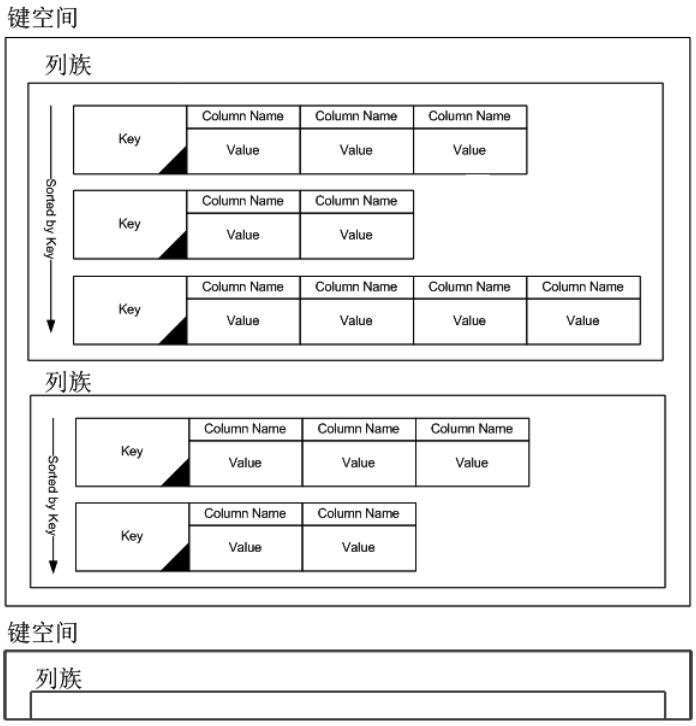


图 4-11　键空间、列族、行、列的关系

所以，Cassandra 中的寻址是一个四维或者五维散列表：

```
[Keyspace][ColumnFamily][Key][Column Name]
```

比如：

```
[gdpi][computer][zsan][age]
```

返回广东理工职业学院（gdpi）计算机系（computer）张三（zsan）这个学生的年龄。

OpenStack 生态中的 HBase 也是列数据库，目前排名第二。

10．CockroachDB

CockroachDB 为谷歌 Cloud Spanner（全球分布式数据库）的开源实现，同时具备 SQL 数据库的强一致性和 NoSQL 的横向扩展性，归属于 NewSQL

大类。CockroachDB 的架构如图 4-12 所示。

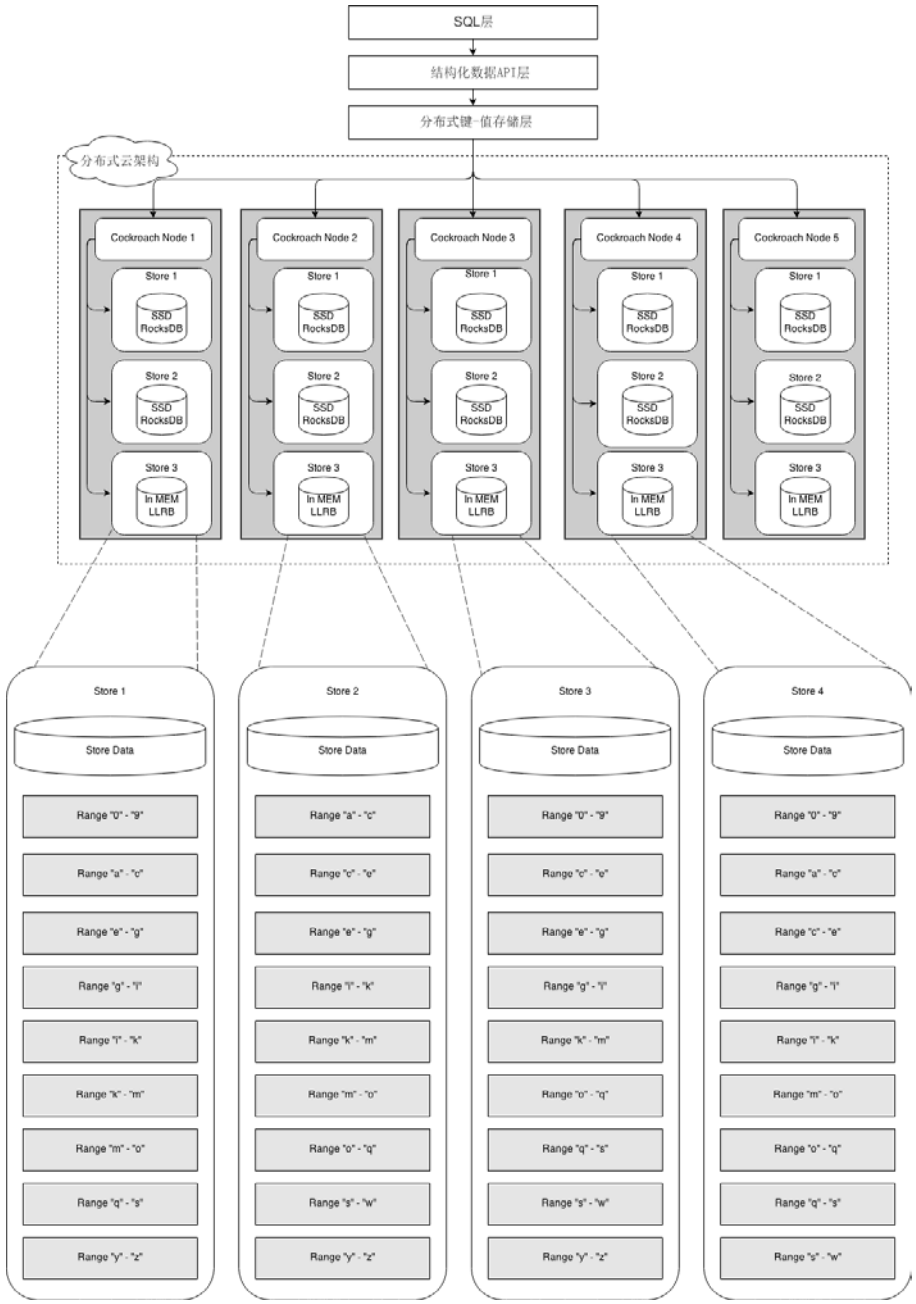


图 4-12　CockroachDB 的架构

SQL 层给用户呈现标准的 SQL 交互接口：库、表、列、索引等，用户的标准 SQL 语句被第二层“结构化数据 API”转换为 NewSQL 特有的操作。数据被分片（range）并以多副本形式保存在多台机器的磁盘中，每片数据对应一个“键-值”对，全部的“键-值”对被保存在第三层“分布式键-值存储层”中。在图 2-12 所示的示例中，以三副本存储数据片，例如“0”至“9”数据片同时保存在第一、三、四号机器中。

4.1.5 中间件

1．Hadoop

当一个大的任务由一台机器在规定的时间内不能完成时，人们就要采用分布式计算，即很多台机器联合起来共同完成任务。换句话说，就是把大任务拆分成许多个小任务，然后再把这些小任务分配给多台计算机去完成。参与计算的多台计算机组成一个分布式系统，需要运行一系列的分布式基础算法。Hadoop 就是一个分布式计算平台，用 Java 语言开发，包含 Common、MapReduce 和 HDFS 三个核心部件（HDFS 和 MapReduce 是最核心的两个部件）。其中，Common 为 Hadoop 的其他项目提供了一些常用工具，主要包括系统配置工具 Configuration、远程过程调用 RPC、序列化机制和 Hadoop 抽象文件系统等；MapReduce 是处理海量数据的计算模型；而 HDFS 用于存储海量数据，它具备高度容错性，能在低成本的通用硬件机器上稳定运行。Hadoop 实现了分布式计算中的基础算法（如一致算法、选举算法、故障检测、快照等），同时为用户提供了编程和命令接口。程序员调用这些函数能轻松写出分布式应用程序，我们都知道，如果一切从头开始，要完成一个分布式程序的编写是异常艰难的。Hadoop 在海量非结构化数据处理方面能充分展示它的优势，如消费者购买行为分析、商品推荐、关键词检索、信贷风险评估等。

如图 4-13 所示，Hadoop 其实就是一个分布式计算平台，它“覆盖”在操作系统之上，向上提供函数调用（API）和命令接口，在水平方向完成分布式系统的基础算法。作为编程人员和用户，只要了解 API 和命令即可。

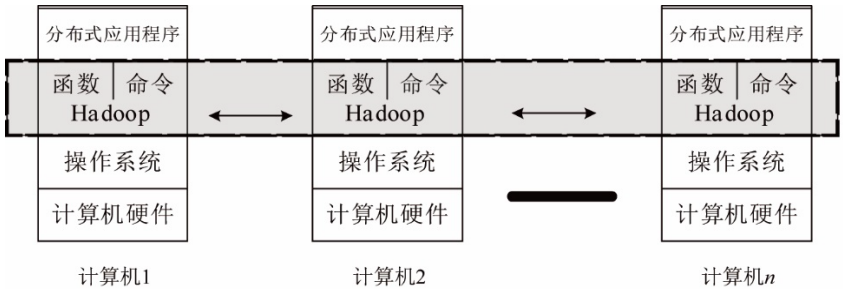


图 4-13　Hadoop

基于 Hadoop 平台衍生出来的开源项目主要有 Yarn、HBase、Hive、ZooKeeper、Avro、Sqoop、Mahout、Crossbow 等。

以 Hadoop 为基础的生态目前已经成为大数据的标准方案，被广泛用于金融、市场、电信、交通等行业的海量数据分析，在即将到来的大数据时代，它将会发挥更大的作用。在中国，很多行业（如银行、电信、移动、电力、石油、交通等）沉淀了大量的业务数据，对这些海量数据进行挖掘和分析，将会带来巨大的价值。

用 Hadoop 构建的应用实例对于计算资源的消耗具备两个明显的特征：一是资源需求大，二是资源需求具备季节性。前者表明 Hadoop 需要大量的存储、计算和网络带宽；后者表明除存储需求是经常性占用外，在运行 MapReduce 时才需要大量的计算和网络资源，而分析大量数据的工作并不是经常性的——称为季节性。因此，云计算是大数据天生的计算资源供应途径，云计算的资源弹性很好地满足了大数据的季节性计算资源需求。也就是说，大数据是云计算经典的应用案例。当然，也可以按照大数据对计算资源的波峰需求静态配给计算资源，但是这种方案会造成资源的巨大浪费。

2．LVS

LVS 是 Linux Virtual Server 的首字母缩写，意为 Linux 虚拟服务器，即把许多台物理 Linux 计算机逻辑上整合成一台超级计算机，对用户来说感觉只有一台计算能力很强的服务器，LVS 原理如图 4-14 所示。LVS 就是一个由软

件实现的负载均衡器，工作在 7 层网络 ISO 模型的 2 至 4 层，是由章文嵩博士开发的，代码已经并入了 Linux 内核。利用它，再加上一台廉价的计算机，就能构建一台企业级的负载均衡器。而那些外国大公司的负载均衡器，售价都要十几万元，甚至几十万元，便宜的也要几万元，LVS 出来后，这些产品都不得不降价。负载均衡器的作用就是把任务分配给最合适的服务器。比如一个大型购物网店，有 100 台同样配置的服务器在运行，如果某一时刻有 10 万用户在线购物，那么通过负载均衡器，每台服务器差不多承担 1000 个在线购物用户。另外，两个较为流行的第七层负载均衡器是 Nginx 和 HAProxy，针对应用做均衡，所以能适应的负载种类没有 LVS 多。

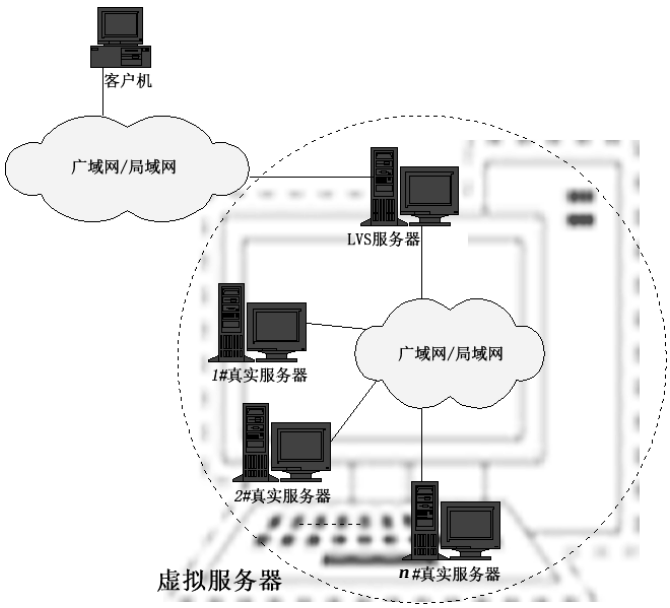


图 4-14　LVS 原理

3．Linux-HA

也许有读者会问：“负载均衡器本身故障怎么办？”是的，如果负载均衡器出现故障，那么整个系统（如网店）将会瘫痪。所以人们开发了各种集群软件，如 Linux-HA 和 Keepalive 等，而微软干脆就在 Windows 服务器版中集成故障转移集群软件。集成故障转移集群软件的核心思想是，实时检测故障

机器并及时让好的机器接管工作，对外提供高可用性。Linux-HA 意为 Linux 高可用性项目，此项目具体包含如下几个组件。

- Heartbeat：负责维护集群中各节点的信息及它们之间的心跳通信。
- Pacemaker：集群资源管理器，是核心组件，客户端通过 Pacemaker 来配置、管理并监控整个集群。OpenStack 高可用性部署实例中一般都采用 Pacemaker 和 HAProxy。
- Resource Agent：用于控制服务启停、监控服务状态的脚本集合，本地资源管理器（LRM）调用这些脚本来启动、停止、监控各种集群资源。
- Cluster Glue：包含一套函数库和工具，在集群栈中，除集群消息传输（由 Heartbeat 承担）、集群资源管理（由 Pacemaker 承担）和资源代理（由 Resource Agent 承担）功能外，其他功能都由 Cluster Glue 来完成。它包含的两个主要部分是 LRM 和 Stonith，前者是本地资源管理器，后者的任务是隔离故障机器。

通过心跳信号（Heartbeat）检测故障，一台好的计算机会不断向其他计算机发送心跳信号，也会接收其他计算机发送过来的心跳信息。当在规定的时间内没有收到对方计算机的心跳信号时，就启动应急预案，进一步确认故障并准备接管那台计算机的任务。

例如，我们采用两台 LVS 计算机，并分别安装和配置 Linux-HA，一台 LVS 计算机作为工作机，另一台作为备份机，两台 LVS 计算机互相监督对方的运行状态。当工作机故障时，备份机接管负载均衡任务并报警；相反，当备份机出故障时，只报警，提醒技术员维修备份机。两台 LVS 计算机同时出故障是比较糟糕的情况，不过这种情况发生的概率很小，除非机房断电或者遭到雷击。对于一些非常关键的应用，可以增加参与负载均衡的服务器数量来提高可靠性，如民航飞机上采用 5 台服务器。

4．静态网站服务器

当我们浏览一家公司的网站时，很可能就是跟那家公司服务器上的

Apache 程序打交道，网页浏览器与 Apache 成了标准的 C/S 模式，浏览器是客户端，而 Apache 是服务端。Apache 首先把主页对应的文件 index.html 发给我们，我们看到主页内容后，单击主页上的某个链接，它又把该链接对应的文件发给我们，访问静态网页的过程如图 4-15 所示。

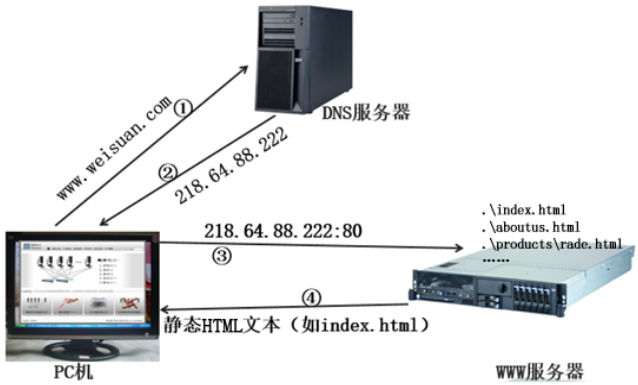


图 4-15　访问静态网页的过程

配合 PHP 引擎，Apache 也支持 PHP 动态网页。过程为：当 Apache 收到用户要浏览的 PHP 文件后，把这个 PHP 文件发给 PHP 执行引擎；PHP 执行引擎执行该 PHP 文件，产生一个临时的静态网页文件并发回给 Apache；最后 Apache 把这个临时的静态网页文件发给用户。采用 Perl、Python 和 Ruby 脚本语言编写的动态网页，其工作过程与 PHP 类似。

5．动态应用服务器

开源的动态应用服务器有 JBoss、Tomcat、Geronimo、JOnAS，关于这些项目更详细的介绍，请参考相应的官方网站。

4.1.6　基础服务组件

基础服务组件包括 DNS、DHCP、用户管理、身份鉴别、权限管理、事后审计、域控等。

1．DNS

DNS 是一个倒树形结构的分布式数据库，里面存储的信息主要有 IP 地址到域名的映射记录、域名到 IP 地址的映射记录、邮件路由记录等。Bind 就是用于实现这个数据库并且允许人们通过域名查询 IP、通过 IP 查询域名或者查询邮件服务器的信息等，我们在配置计算机网卡参数时要指定多个 DNS 服务器 IP 地址。在中国，这些 DNS 服务器大部分都在电信机房，如果不设置或者 IP 地址设置错误，则计算机无法上网。不过，建议大家使用谷歌提供的 DNS 服务器，IP 地址是 8.8.8.8 和 8.8.4.4，速度快又方便记忆。

BIND（Berkeley Internet Name Domain）是一款开源的 DNS 服务器软件，是目前互联网上使用最广泛的 DNS 服务器软件，能在各种流行的操作系统上运行。现由互联网系统协会负责开发与维护，可从该组织的官方网站下载源代码，然后编译安装，目前稳定版本是 9.11.0。

2．DHCP

DHCP（Dynamic Host Configuration Protocol，动态主机配置协议）允许一台计算机通过 DHCP 服务器自动获取网络配置参数，如网卡的 IP 地址、网关、掩码和主机名字等，这样只要启动计算机就能上网，而不用手工配置计算机的网络参数。DHCP 服务器的第二个功能是定义一些与自动安装操作系统有关的参数，比如一台计算机设置从网卡启动，只要 DHCP 服务器配置了 PXE 启动有关的参数，那么用户的计算机就能自动根据配置参数来启动或者安装操作系统，而不管原来有没有安装操作系统。这一点在可伸缩云端和零存储云终端的环境下非常有用，服务器随时加入随时启用，云终端启动就自动下载统一的操作系统内核并运行统一的登录程序。

dhcp3-server 和 isc-dhcp-server 是目前比较流行的开源 DHCP 软件项目，前者已经被广泛使用；后者发展迅速，操作系统 Ubuntu 12.04 版本开始使用它了。isc-dhcp-server 也是由互联网系统协会负责开发与维护的，可从该组织的官方网站 http://www.isc.org/下载源代码，然后编译安装，目前稳定版本是 4.3.5。

3．Kerberos

Kerberos 这一名词来源于希腊神话“三个头的狗——地狱之门守护者”。但在 IT 领域，Kerberos 是指一种计算机网络上的身份认证协议，主要用来在非安全的计算机网络中，对个人通信以安全的手段进行身份认证。Kerberos 又指麻省理工学院为这个协议开发的一套计算机软件，软件设计上采用客户机/服务器结构，并且能够进行相互认证，即客户端和服务器端均可对对方进行身份认证，可以用于防止窃听、防止重放攻击、保护数据完整性等场合，是一种应用对称密钥机制进行密钥管理的系统。Kerberos 的扩展产品也使用公开密钥加密方法进行认证。

当有 *N* 个人使用该系统时，为确保在任意两个人之间进行秘密对话，系统至少保存其与每个人的共享密钥，所需的最少会话密钥数为 *N* 个。Kerberos 身份认证流程如图 4-16 所示。

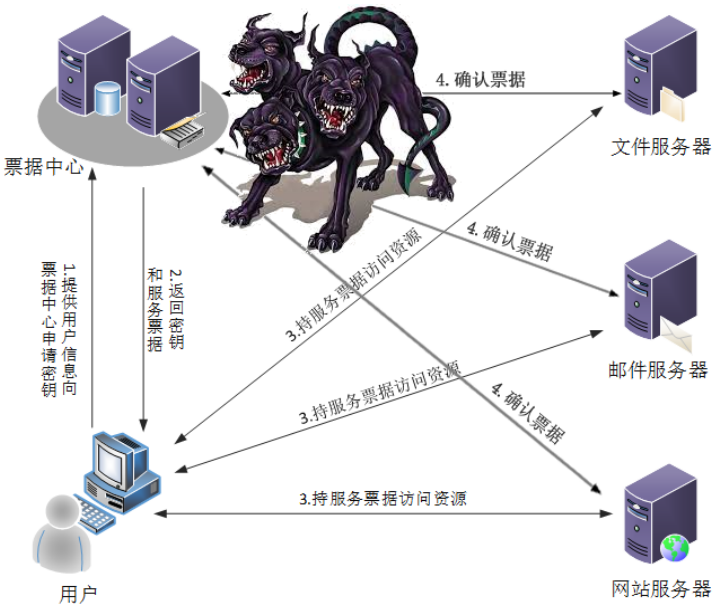


图 4-16　Kerberos 身份认证流程

如果想进一步了解 Kerberos 的原理，请在网上搜索“Kerberos 原理对话”，其以四幕话剧的形式来阐述 Kerberos 的原理，非常经典。

Kerberos 可用来部署单点登录环境——只要记住一个密码就能登录到任何一个应用，类似购买景点通票。微软的 AD 域就采用了 Kerberos。另外，图 4-17 所示是笔者为国内一家大型企业部署的单点登录草图，实现了企业内部上千台 Linux 服务器的单点登录。

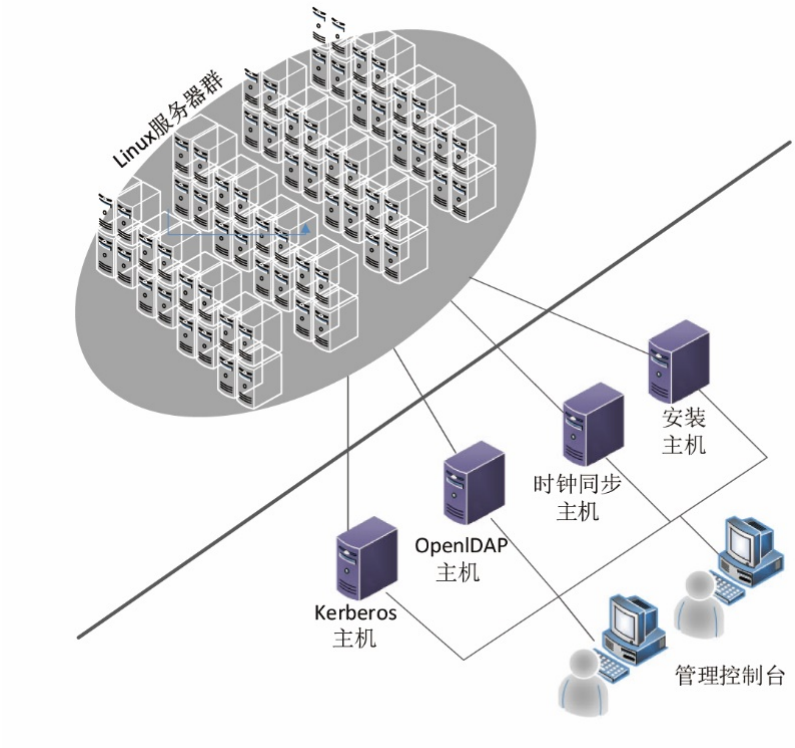


图 4-17　某大型企业部署的单点登录草图

另一套采用 Kerberos 实现身份认证和权限管理的开源软件是 OpenAM/OpenDJ/OpenIDM，由 ForgeRock 公司开发和维护。

OpenAM 提供集中认证、授权、单点登录，以及与其他单位建立联邦治理的全面解决方案，它是完全开源的，其前身是 Sun 公司的 OpenSSO。OpenAM 具有高度可扩展的、模块化的、易于部署的架构，具体包括如下内容。

- 身份认证：允许通过使用者的周边信息实现动态的身份识别，使用者的周边信息包括地理位置、时区、IP 地址、接入设备等，当然也支持密码认证。身份认证模块相当于门卫，控制哪些人可以进入，哪些人不可以进入，至于进去后能做什么事情，它不会加以限制。
- 权限管理：根据事先定义好的权限和策略方案来控制用户登录后的行为，即规定他能做什么，不能做什么。
- 联邦和单点登录：每个用户只需一个账户和密码，就可以访问众多的应用系统和网络资源。想必大多数人都有过这样的痛苦经历：要记住众多的用户名和密码，以便访问许多系统，如邮件系统、工作网站、论坛、微博等，单点登录彻底解决了这个问题。
- 外围系统认证集成：允许集成认证目前流行的社交网站，如推特、Facebook、微博等。
- 自适应风险评估：根据登录用户的操作行为来评估其动机，如果怀疑其动机不纯，那么自动提升身份级别鉴别，要求提供额外的身份信息，以便进一步确认其真实身份。

OpenDJ 是一款遵循 LDAPv3 协议的目录服务器，采用 Java 语言开发，完全开源。它的前身是 Sun 公司的 OpenDS 项目，现在由 ForgeRock 公司接手。用 OpenDJ 可以搭建高性能、高可用性和高安全性的企业级目录服务系统，处理上千万条数据记录，响应时间均在毫秒级以内。

OpenDJ 允许多主/复制部署（Multi-Master），数据可以集中式存储，也可分布式存放，这一点要优于另一个目录服务开源项目——OpenLDAP。另外，OpenDJ 消耗很少的计算机资源（CPU、内存等），由于是纯 Java 开发，所以 OpenDJ 可以在任何平台上运行。

利用 OpenIDM，企业可以轻松集中管理用户信息，包括用户账号和权限管理，适合在企业级、云端和移动社交网络上使用。

另外，耶鲁大学开发的 CAS 项目也是非常有名的认证系统，官方网站为 https://www.apereo.org/projects/cas/。

Keycloak 是一个完全开源的用户身份和访问管理工具，支持 OpenID Connect、OAuth 2.0、SAML 2.0 等标准协议，采用 Java 开发。

4．Radius

Radius 是指局域网接入认证，用于解决一台计算机需要认证才能接入和使用局域网的问题。比如一个非本公司的外人带了笔记本电脑进来，他找到一根网线插入笔记本电脑，如果他能使用公司内部的局域网（不管是自动分配 IP 还是手动配置网卡参数），那么这时公司内部的很多资源就面临危险，因为一个恶意的人可以做网络窥探、运行额外的 DHCP 使网络紊乱、广播病毒、端口扫描、消耗网络带宽等操作。有了局域网接入认证系统后，只有合法的设备（通常捆绑了网卡的 Mac 地址）才能接入局域网，没有事先登记的设备被拒绝接入，即使配好了网卡参数。

FreeRadius 是一款开源的实现局域网接入认证的软件，可从网站 http://freeradius.org/下载源代码，目前稳定的版本是 3.0.17。具体的 FreeRadius 认证过程如图 4-18 所示。

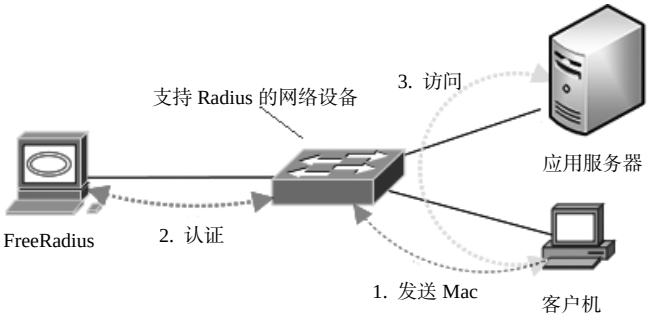


图 4-18　FreeRadius 认证过程

4.1.7 云管理工具

首先我们看看一个基于虚拟机的 IaaS 云端的形成过程。

虚拟机要运行在虚拟机软件里，而虚拟机软件运行在物理机上。一台物理机通过虚拟机软件可以虚拟出多个虚拟机，虚拟机里再安装 Windows 或者 Linux 操作系统及各种应用软件，然后用户通过远程桌面等方式连接到虚拟机并使用虚拟机中的应用软件。如果一个云中心只有一台物理机，那么只要一个虚拟机软件即可，至此，一个最小的准云端系统就创建完成了。

接下来继续增加物理机，每台物理机上运行多个虚拟机，这样云端虚拟机的数量就增加了很多。但是问题来了，如何避免因为一些物理机故障导致部分云终端用户不能使用虚拟机的情况？为了解决这个问题，人们引入了集群技术，允许虚拟机在集群中的任何一台机器上运行，这样故障机器上的虚拟机就能“漂移”到其他物理机上并继续运行。但这样又产生了一个新的问题，那就是如何保存虚拟机本身（虚拟机就是对应宿主机上的若干文件）才能确保“漂移”动作既快速又正确地完成呢？对此，人们又引入了中央存储技术，即把全部的虚拟机镜像文件保存在中央存储设备上，让集群里的物理机都能共享访问。这样，一个由多台物理机组成集群、由多个集群组成的云端系统雏形就形成了。

然后我们又会自然而然地想到其他问题：第一，如何管理云端众多的虚拟机？于是人们开发了云端管理工具，采用此工具可以很轻松地创建、删除、迁移、启动、关闭、冻结和备份虚拟机。第二，如何给多台虚拟机组建网络？为此，人们又发明了虚拟网卡、虚拟交换机、网络功能虚拟化和软件定义网络技术。

云端最核心的部分就是虚拟化软件、中央存储设备和虚拟机管理工具（有的人喜欢称云管理平台，甚至直接称云计算操作系统）。称其为“云计算操作系统”是有一定道理的，与计算机操作系统（如 Windows、Linux 等）类似，计算机操作系统的核心功能是管理进程，只有进程才能申请资源（CPU、内

存、打印机等）；而在云端，虚拟机是申请资源（计算、存储、网络等）的基本主体，管理虚拟机是云计算操作系统最核心的功能。但是云计算操作系统远没有计算机操作系统那么复杂，称其为操作系统有点夸大其词。

云管理工具分为 IaaS、PaaS 和 SaaS 三种类别，SaaS 类别的云管理工具与网店差不多，主要是租户注册、自助、购买、结算等，与业务关联性大，所以目前没有统一的标准版产品。比如，针对一款大型可靠性软件系统的云化项目，笔者设计的 SaaS 云管理软件包含两大部分：一是 SaaS 业务门户，二是 SaaS 管理门户，其结构分别如图 4-19 和图 4-20 所示。

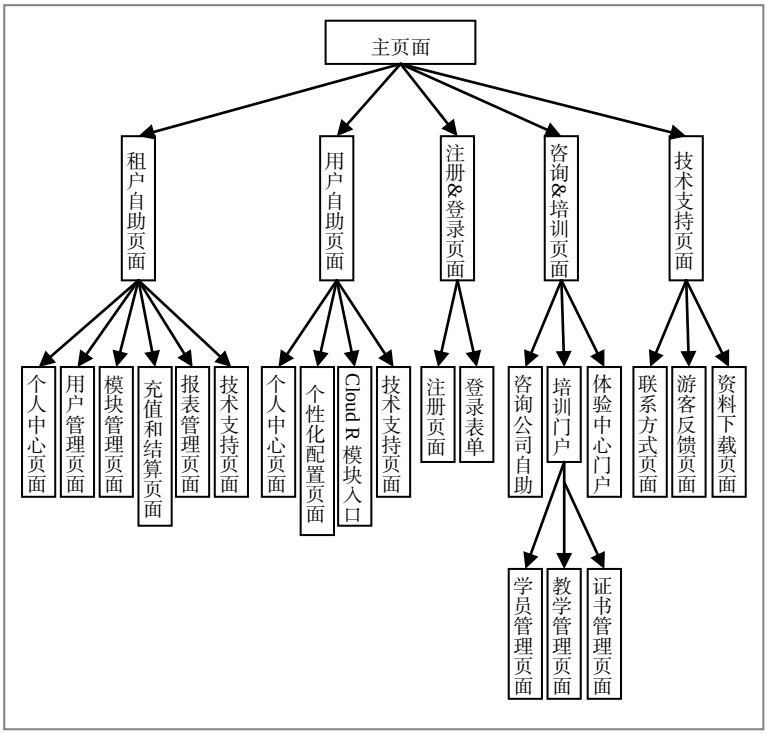


图 4-19　SaaS 业务门户结构

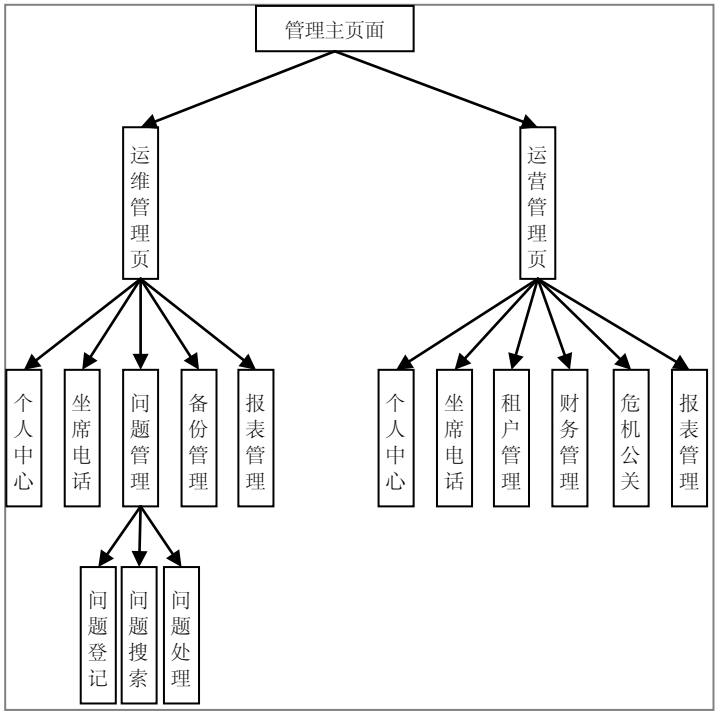


图 4-20　SaaS 管理门户结构

尽管框架类似，但由于 SaaS 云计算与具体的应用软件紧密相关，所以截至目前，还没有出现通用的开源或商业化的 SaaS 云计算管理工具。目前的 SaaS 云计算提供商都是自己开发自己使用。PaaS 管理工具有红帽公司的 OpenShift 和 Cloud Foundry 社区的 CloudFoundry，两者都是开源的，在 https://financesonline.com/网站上，OpenShift 的得分要高于 CloudFoundry。下面重点介绍一些 IaaS 管理工具。

1．OpenStack/RedHat OpenStack Platform

OpenStack 是当今最具影响力的云计算管理工具——通过命令或者基于 Web 的可视化控制面板来管理 IaaS 云端的资源池（服务器、存储和网络）。它最先由美国国家航空航天局（NASA）和 Rackspace 在 2010 年合作研发，现在参与的人员和组织汇集了来自 100 多个国家的超过 9500 名的个人和 850 多个世界上赫赫有名的企业，如 NASA、谷歌、惠普、Intel、IBM、微软等。

OpenStack 系统或其演变版本目前被广泛应用在各行各业，包括自建私有云、公有云、租赁私有云及公私混合云，用户包括思科、贝宝（Paypal）、英特尔、IBM、99Cloud、希捷等。

OpenStack 支持 KVM、Xen、Lvc、Docker 等虚拟机软件或容器，默认为 KVM。通过安装驱动，也支持 Hyper-V 和 VMware ESXi，不过有些功能暂时不支持，具体的虚拟机管理器支持矩阵参见相关网站。

OpenStack 采用 Python 语言开发，遵循 Apache 开源协议，因此相比 CloudStack 来说，更轻量化，效率更高。OpenStack 每半年发行一个新版本，截至发稿前最新版本是第十七版本 Queens，不同于其他软件的版本号采用数字编码，OpenStack 采用一个单词来描述不同的版本，其中单词首字母指明版本的新旧。比如目前的版本 Queens 就比之前的 Pike 要新，同时“Q”在 26 个字母中排行第十七，所以称第十七版本。但是不建议在生产环境中使用最新版本，因为最新版本还在不断地打补丁。

围绕 OpenStack 发展起来的企业很多，为客户提供 OpenStack 实施、培训、运维、定制等业务，之前企业总是或多或少地加入自己的一些封闭技术，从而导致 OpenStack 的互操作性受损。为此，2015 年 OpenStack 基金会在温哥华峰会上正式推出互操作性认证，通过认证的产品被贴上“OpenStack Powered”标识。虽然第一批只有 14 家厂商经过认证测试，但这却是一个重要的里程碑事件，基金会已经拿出足够的诚意来解决问题，并且众多厂商也开始真正跟进。对用户而言，选择经过认证的云服务提供商，能够实现在不同 OpenStack 云计算之间的自由迁移。

OpenStack 生态系统已从“孵化/集成”模式转移到“大帐篷”模式，在此模式下，既保持了对规模较小的核心项目的关注，又积极鼓励在更广泛的主流生态环境中的自由创新，而以前的“孵化/集成”模式只是把孵化成功的项目集成到主流生态中。“大帐篷”模式把 OpenStack 的组件进行分类，目前包括 6 个核心组件（Nova、Neutron、Swift、Cinder、Keystone、Glance）和 14 个可选组件，每个组件包含若干个服务，后续版本中组件分类及数量都

可能会发生变化，如图 4-21 所示。

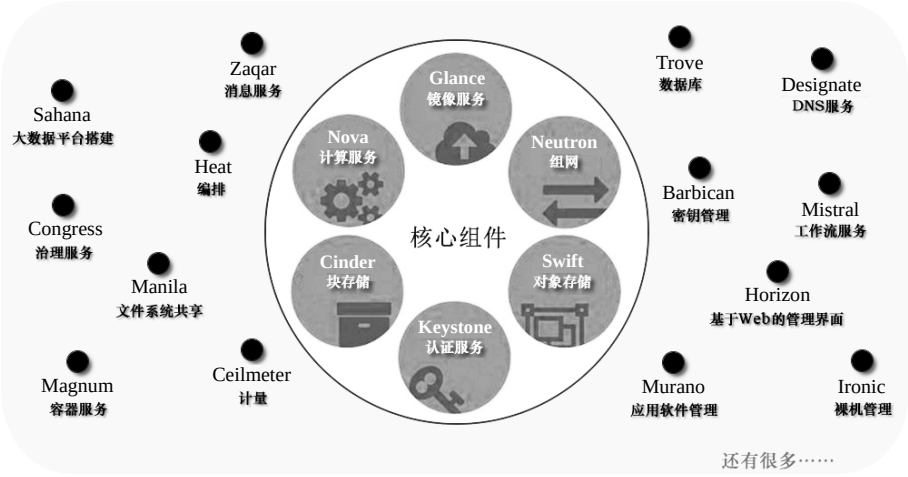


图 4-21 “大帐篷”模式下的组件

表 4-2 列出了 Queens 版本中各个组件的功能介绍。

表 4-2 Queens 版本中各个组件的功能介绍

分类	组件名称	功　能
核心组件	Nova	管理虚拟机的整个生命周期：创建、运行、挂起、调度、关闭、销毁等。这是真正的执行部件，接受 DashBoard 发送的命令并完成具体的动作。但是 Nova 不是虚拟机软件，所以还需要虚拟机软件（如 KVM、Xen、Hyper-v 等）配合
	Neutron	管理网络资源，提供了一组应用编程接口（API），用户可以调用它们来定义网络（如 VLAN），并把定义好的网络附加给租户。Networking 是一个插件式结构，支持当前主流的网络设备和最新网络技术
	Swift	是 NoSQL 数据库，类似 HBase，为虚拟机提供非结构化数据存储，它把相同的数据存储在多台计算机上，以确保数据不会丢失。用户可通过 RESTful 和 HTTP 类型的 API 来和它通信。这是实际的存储项目，类似 Ceph，不过在 OpenStack 具体实施时，人们更愿意采用 Ceph
核心组件	Cinder	管理块设备，为虚拟机管理 SAN 设备源。但是它本身不是块设备源，需要一个存储后端来提供实际的块设备源（如 iSCSI、FC 等）。Cinder 相当于一个管家，当虚拟机需要块设备时，询问管家去哪里获取具体的块设备。它也是插件式的，安装在具体的 SAN 设备里。

续表

分类	组件名称	功　能
核心组件	Keystone	为其他服务提供身份验证、权限管理、令牌管理及服务名册管理。要使用云计算的所有用户事先需要在 Keystone 中建立账号和密码，并定义权限（注意：这里的“用户”不是指虚拟机里的系统账户，如 Windows 7 中的 Administrator）。另外，OpenStack 服务（如 Nova、Neutron、Swift、Cinder 等）也要在里面注册，并且登记具体的 API，Keystone 本身也要注册和登记 API
	Glance	存取虚拟机磁盘镜像文件，Compute 服务在启动虚拟机时需要从这里获取镜像文件。这个组件不同于上面的 Swift 和 Cinder，这两者提供的存储是在虚拟机里使用的
可选组件	Horizon	提供了一个网页界面，用户登录后可以做这些操作：管理虚拟机、配置权限、分配 IP 地址、创建租户和用户等。本质上就是通过图形化的操作界面控制其他服务（如 Compute、Networking 等）。当然，如果你熟悉命令，也可以直接采用命令来完成相应的任务
	Heat	如果要在成千上万个虚拟机里安装和配置同一个软件，该怎么办？采用 Orchestrates 是一个不错的主意，它向每个虚拟机里注入一个名叫 heat-cfntools 的客户端工具，然后就能同时操作很多虚拟机
	Sahana	使用户能够在 OpenStack 平台上（利用虚拟机）一键式创建和管理 Hadoop 集群，实现类似 AWS 的 EMR（Elastic MapReduce Service）功能。用户只需要提供简单的配置参数和模板，如版本信息（CDH 版本）、集群拓扑（几个 Slave、几个 DataNode）、节点配置信息（CPU、内存）等，Sahara 服务就能够在几分钟内根据提供的模板快速部署 Hadoop、Spark 及 Storm 集群。Sahana 是一个大数据分析项目
	Ironic	把裸金属机器（与虚拟机相对）加入资源池中
	Zaqar	Zaqar 为 Web 和移动开发者提供多租户云消息和通知服务，开发人员可以通过 REST API 在其云应用的不同组件中通过不同的通信模式（如生产者/消费者或发布者/订阅者）来传递消息
	Ceilometer	结合 Aodh、CloudKitty 两个组件，完成计费任务，如结算、消耗的资源统计、性能监控等。OpenStack 之所以能管理公有云，一是因为 Ceilometer 的存在，二是因为引入了租户的概念
	Barbican	是 OpenStack 的密钥管理组件，其他组件可以调用 Barbican 对外暴露的 REST API 来存储和访问密钥
	Manila	为虚拟机提供文件共享服务，不过需要存储后端的配合
	其他组件：Congress（策略服务）、Designate（DNS 服务）、Freezer（备份及还原服务）、Magnum（容器支持）、Mistral（工作流服务）、Monasca（监控服务）、Searchlight（索引和搜索）、Senlin（集群服务）、Solum（APP 集成开发平台）、Tacker（网络功能虚拟化）、Trove（数据库服务）	

OpenStack 各个组件的关系如图 4-22 所示。

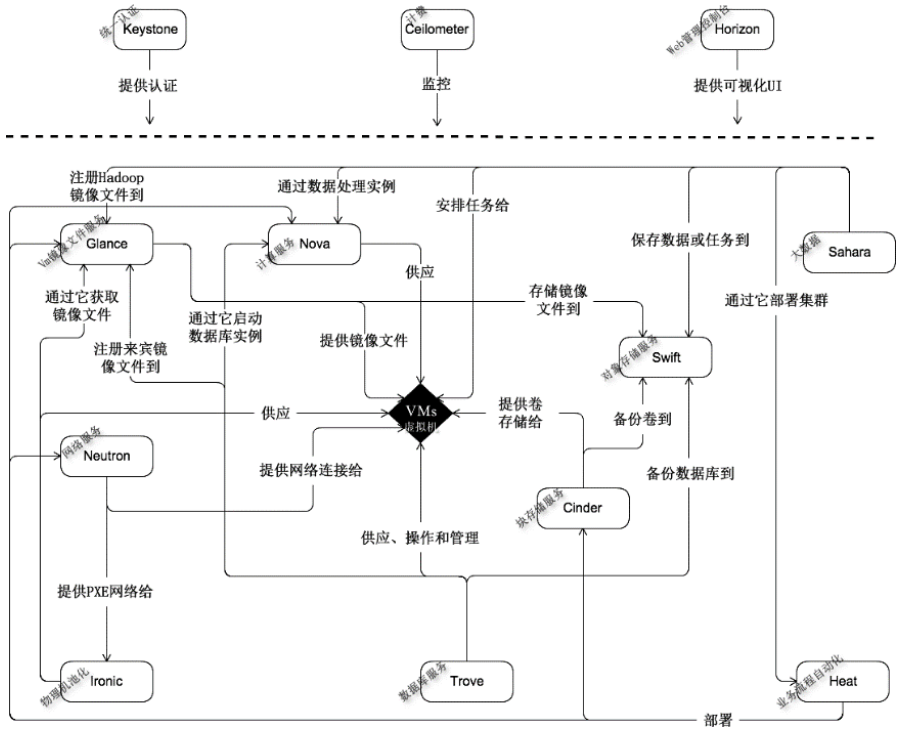


图 4-22　OpenStack 各个组件的关系

OpenStack 的组件众多，根据云端的实施过程，再结合图 4-22，我们来梳理一下各个组件的作用：云端要运行很多虚拟机，所以需要在很多服务器中安装并运行虚拟机软件（如 KVM、Xen 等），有的客户为了安全起见，愿意出高价直接租赁物理机（裸金属机器），所以要用 Ironic 组件来池化物理机，以便用户能远程使用。这些运行了虚拟机软件的服务器和被池化的物理机统称为计算节点。为了让 Horizon 组件以可视化的 Web 页面来统一操纵计算节点上的虚拟机，需要在计算节点上安装 Nova 组件，Nova 组件还与其他组件打交道。为了让一台虚拟机能在集群内的任一计算节点上快速漂移，虚拟机对应的镜像文件必须存放在共享场所，到底存放在哪里，由 Glance 组件指定。例如在图 4-22 中，由 Glance 指定存放在 Swift 组件内，在实际的实施案例中，

也可以存放在 Ceph 中。虚拟机之间需要联网，由 Neutron 组件负责。虚拟机里面可能还要使用块设备（如硬盘），这需要 Cinder 组件的配合；虚拟机里可能需要用到共享文件服务，由 Manila 组件提供服务。云端的计算节点很多（如 1000 台），所以虚拟机就更多（如 10 万台），如果要给它们统一安装一个软件或配置某项参数，那么是不是需要手工一台一台操作呢？显然，手工操作费时费力，而且容易出错，有了 Heat 组件，我们就可以轻松完成这个任务。OpenStack 的各个组件都是对外暴露 REST API 接口的，以便其他程序调用，调用时都要进行身份验证和权限管理，这由 Keystone 组件完成。跟踪用户消耗的资源并计费的任务由 Ceilometer 组件完成（需要 Aodh 和 CloudKitty 组件的配合）。对于 OpenStack 管理的 IaaS 云服务，有人想在上面部署 Hadoop 大数据分析系统怎么办？这时 Sahala 组件可以帮上忙。各组件之间需要通过消息互相联络，所以 Zaqar 和 RabbitMQ 就派上用场了。另外，很多组件需要在数据库中保存配置数据，所以需要用到数据库管理系统（如 MySQL）。

OpenStack 组件的主要作用是充当“中间人”，它不执行具体的实际任务，而由各种第三方软件来完成，比如虚拟机软件由 KVM 承担，网站任务由 Apache 承担，虚拟网络任务由 iptables、DNSmasq、Linux vSwitch、Linux 网桥承担或者统一由 OpenContrail 承担，结构化数据存储任务由 MySQL 或者 PostgreSQL 承担，中央存储任务由 Ceph 承担（也可采用其他产品）。当然，OpenStack 中也有实现具体功能的组件，比如 Swift 做中央存储，我们也可以选择相对发展多年并且被大量使用的第三方产品，如 Ceph。

一个云端往往包含成千上万台服务器，而且还可能分布在世界各地，分别服务符合延时半径范围内的用户。OpenStack 中的“地区”（Region）就是对应地理位置不同的分中心，如中国北京、美国纽约、英国伦敦等。在同一个 Region 中，还可能包含成千上万台机器，如果用一套 OpenStack 中的组件来管理，势必会导致这些组件本身成为瓶颈（随着集群规模的不断增大，消息系统和数据库系统很可能最先成为瓶颈），所以人们又引入了 Cell 功能，以便增强 OpenStack 集群的扩展性，即把一个 Region 划分成多个 Cell，这些 Cell

组成树形结构，父 Cell 主要用于服务通信，它不包含计算节点，子 Cell 具有自己的消息队列、数据库和 Noval-cell 服务。Nova-cell 在 OpenStack 的 Newton 版本中将成为默认项，之前的版本是可选项。在创建虚拟机时，为了规定它能在哪些计算节点集上运行，人们又提出了两个概念，即“可用域”（Availability Zones，AZ）和“主机集”（Host Aggregates，HA），前者可以看成后者的一个特例。“主机集”其实就是根据计算节点的某些属性对计算节点进行逻辑分组的方法，比如可以分成如下几个“主机集”：万兆网卡的机器、拥有两路 CPU 的机器、惠普机器、自组装的机器、A 机柜里的机器、由 UPS 供电的机器等。然后我们创建一台虚拟机，指明在上海云端分部的惠普机器上运行，这样只要全部的惠普机器不同时坏，那么虚拟机就能一直正常运行（但每一时刻只能在一台机器上运行，只有当运行的那台机器出故障时，才会“漂移”到其他惠普机器上继续运行）。“可用域”是用户可见的，用户把自己的多个虚拟机分散到不同的“可用域”中，是为了降低所有虚拟机同时不可用的概率，而“主机集”是管理员可见的，目的是用来隔离虚拟机，从而降低一些特定虚拟机的运行行为对其他虚拟机产生的影响。Region、Cell（第 2 版本）、AZ、HA 的关系如图 4-23 所示。

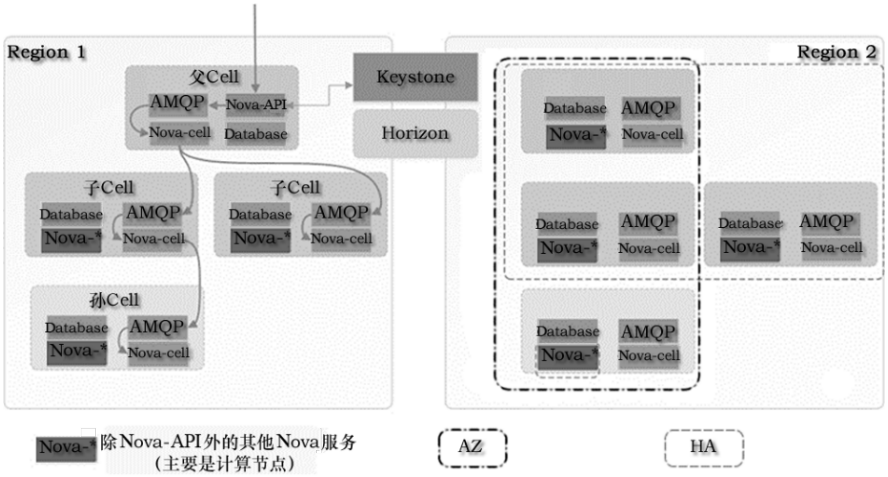


图 4-23　Region、Cell（第 2 版本）、AZ、HA 的关系

从图 4-23 中可以看出，多个 Region 允许共享 Keystone 和 Horizon 服务，也可以完全独立。HA 可以跨 Cell，但是不能跨越 Region，一台机器可以同时属于多个 HA，因为 AZ 是 HA 的特例，所以一台机器允许同时属于 AZ 和 HA。当一个创建虚拟机的请求到达父 Cell 的 Nova-API 时，父 Cell 会通过 Nova-cell 向各个子孙 Nova-cell 广播请求，并一次性决定在哪个子孙 Cell 中的哪台计算节点上创建虚拟机。

在具体部署 OpenStack 时应该遵循“逐步扩展部署法”，如图 4-24 所示。

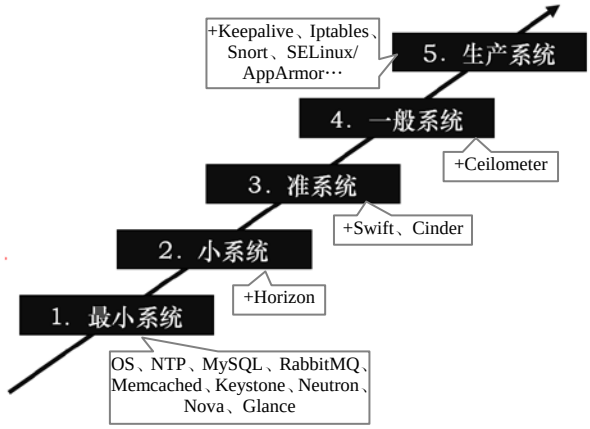


图 4-24　OpenStack 逐步扩展部署法

最小系统具备基本的 IaaS 功能，能通过命令来进行管理，这一步只需安装 OpenStack 的 Keystone、Neutron、Nova 和 Glance 四个组件；此后再安装 Horizon 就成了小系统，这时可通过 Web 图形化界面来执行管理；继续安装 Swift 和 Cinder 就成了准系统，这时能给虚拟机附加磁盘块设备，并能满足大规模的存储需求；再加上计费组件 Ceilometer，就上升为一般系统，一般系统具备公有 IaaS 的功能。但是由一般系统跨到生产系统，需要完成的工作就特别多，其中性能和安全是两个不得不面对的棘手问题。图 2-24 中标注的 Iptables（设立门卫）、SELinux 或 AppArmor（加固系统）和 Snort（巡逻）都是为了强化安全。性能和安全涉及的知识太多，这里不再展开讨论。图 4-25 取自网上，主要考量了安全当中的可用性，供大家参考。

不同于 OpenStack 用于管理虚拟机，由谷歌主推的开源项目 Kubernetes 用于管理容器，不过 OpenStack 生态中的 Magnum 项目也用于容器管理。

2．Apache CloudStack/CloudPlatform

思杰（Citrix）收购 CloudStack 后让其开源，并捐献给 Apache 基金会，让世界上许许多多的开源软件爱好者参与开发、测试和讨论中。当社区版趋于稳定时，思杰公司及时把它转化为服务收费版 CloudPlatform。2016 年 1 月思杰把 CloudPlatform 出售给了 Accelerite 公司，自己又重回 OpenStack 的怀抱，曾经离开时思杰痛陈 OpenStack 的数条罪状，至今让人记忆犹新。采用 CloudStack 或 CloudPlatform 管理的云端逻辑架构，如图 4-26 所示。

从图 4-26 可知，整个云端包含若干个 Region（地区，可以理解为地理位置上的一个城市、一个省份等），默认情况下只有一个 Region。每个 Region 包含多个 Zone（大致相当于数据中心，也允许一个实际的数据中心划分为多个 Zone），每个 Zone 又包含若干个 Pod（相当于机柜，一个 Pod 中的机器位于同一个子网中）和若干个二级存储，每个 Pod 又包含若干个 Cluster（集群），每个集群又包含若干台 Host（服务器）和若干主存储。一个集群不能跨越多个 Pod。每个 Zone 的 Secondary Storage（二级存储）存放与该 Zone 相关的虚拟机模板、操作系统的 ISO 安装镜像文件和磁盘卷快照。Cluster 中的 Primary Storage（主存储）存放虚拟机镜像文件，这些虚拟机可在集群中的任一台 Host 上运行，当一台 Host 发生故障时，其上的虚拟机将被转移到同一集群中的其他 Host 上，并一直保持运行状态，这叫作热迁移；相反，冷迁移要求先关闭虚拟机，然后再在其他计算机启动虚拟机，所以在冷迁移过程中，用户是无法使用虚拟机的。

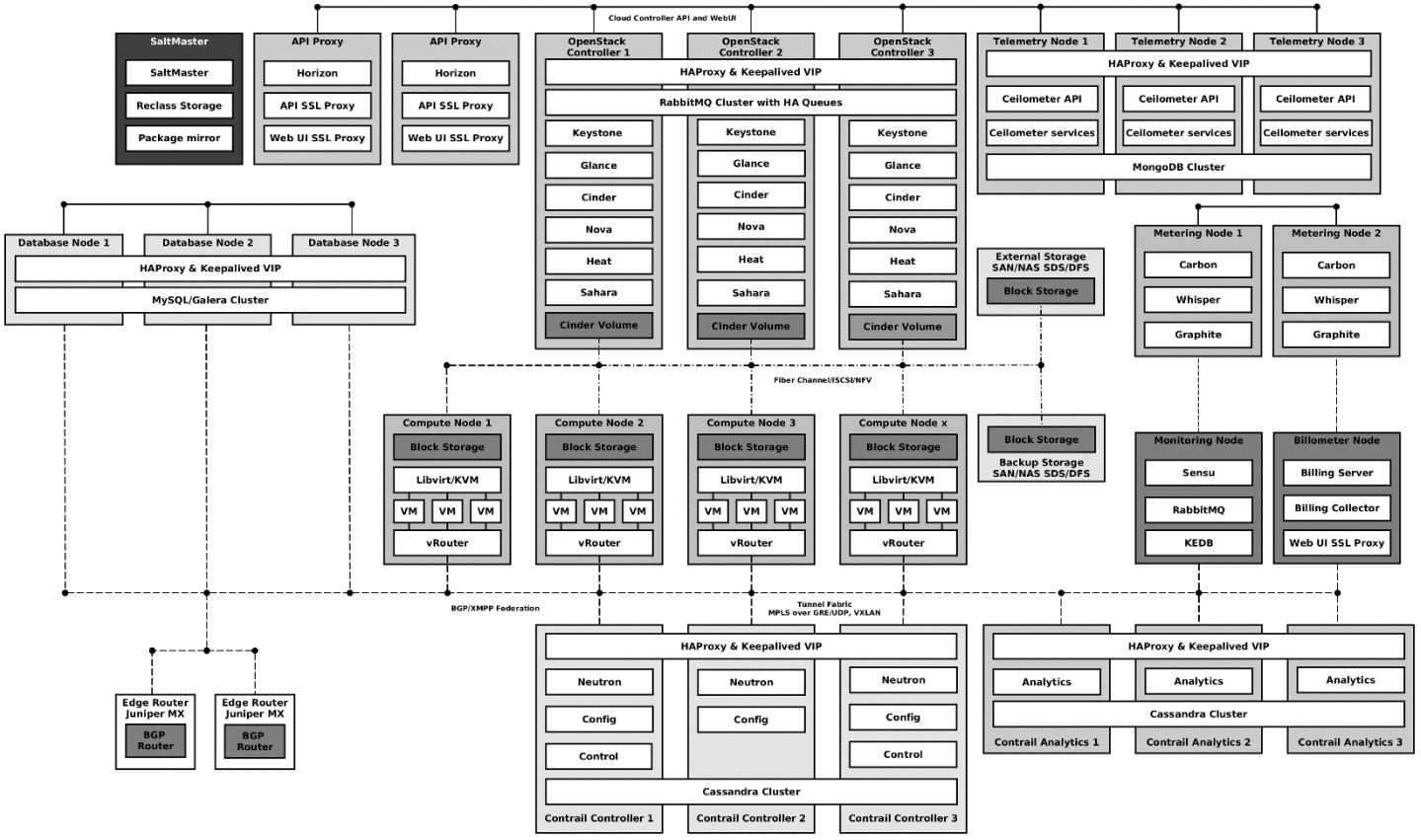


图 4-25　OpenStack 高可用性部署架构

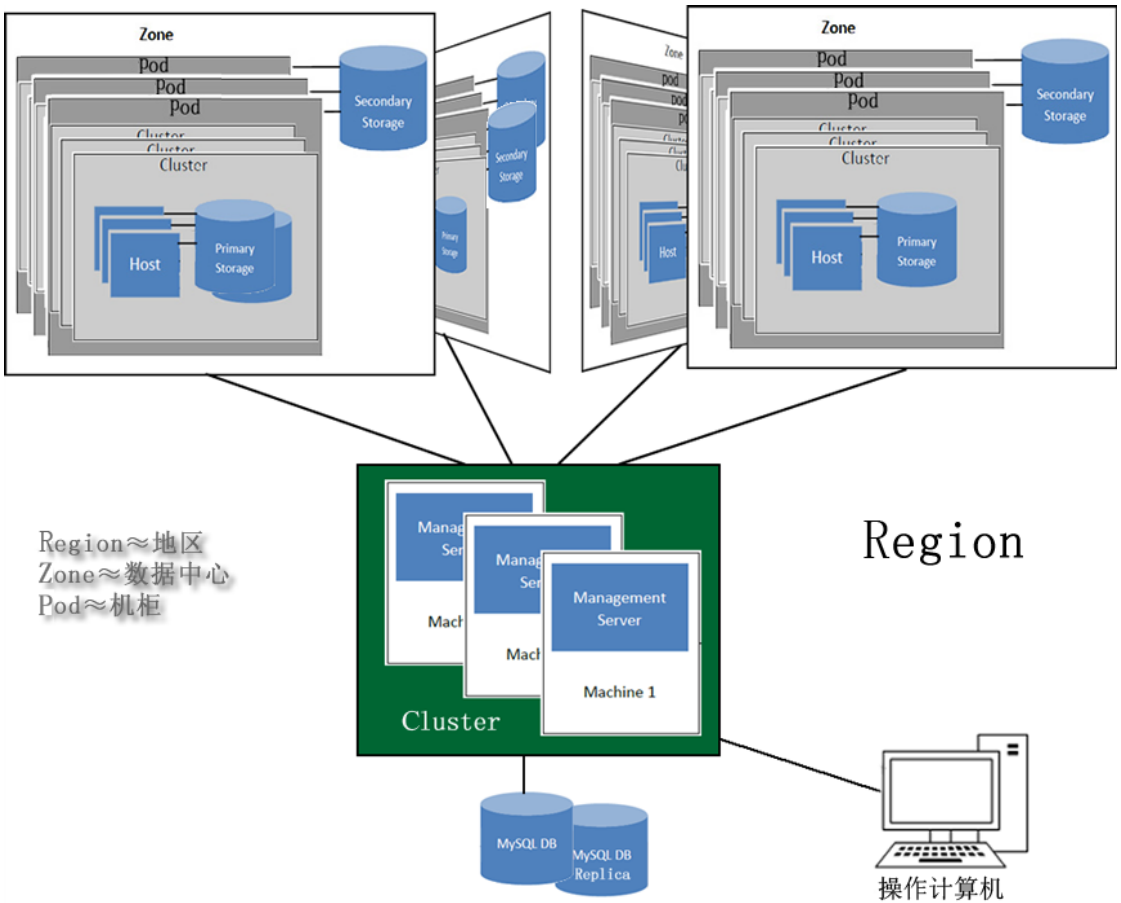


图 4-26　CloudPlatform 管理的云端逻辑架构

CloudStack 或 CloudPlarform 安装在 Management Server（管理服务器）上，其能管理的虚拟机软件有：Xen、RedHat KVM、Citrix XenServer、VMware vSphere、Oracle VM（OVM）等，可以说绝大部分软件都支持。允许多台 Management Server 机器组成管理集群，当然也可以用一台，但是一台可靠性很差，会形成单点故障。CloudStack 采用 Java 语言开发，采用 Tomcat 做应用服务器，采用 MySQL 开源数据库存储配置信息。CloudPlatform 自带了一个安装脚本，利用它安装云端管理工具比较方便，即通过“点菜（单）”方式。云计算管理工程师通过工作台式机或者笔记本、平板电脑使用网站浏览器打开管理服务器上的 Web 页面：http://<管理服务器 IP 地址>:8080/client，即可登录管理页面。关于 Apache CloudStack 中各种概念更详细的介绍，请参见其官方网站 http://docs.cloudstack.apache.org/en/latest/concepts.html。

CloudPlatform 能管理私有云、公有云和公私混合云，在生产环境中，建议采用 CloudPlatform，而不要采用 CloudStack。

3．Oracle VM Manager

在 IT 行业，Oracle 公司具有举足轻重的地位，Oracle 公司发行的虚拟化管理工具 Oracle VM Manager，采用 Java 语言开发，图 4-27 所示是 Oracle 公司官方给出的虚拟环境架构图。严格意义上讲，Oracle VM Manager 只是虚拟机管理工具，而不是云管理工具。

从图 4-27 可以看出，云端由若干服务器池和一个外部存储组成，每个服务器池包含若干台服务器（称为计算节点）。服务器池可以是集群，也可以不是集群。服务器池中的计算节点上安装 Oracle VM Server 虚拟软件（基于 Xen 虚拟化技术），然后运行若干个 dom。其中 dom0 为管理 dom，里面安装和运行 Oracle VM Agent，dom0 有权限访问硬件和驱动，并且负责与 OracleVM 管理器通信。其他 dom 统称为 domU，为用户使用的普通虚拟机，由 dom0 启动并管理。

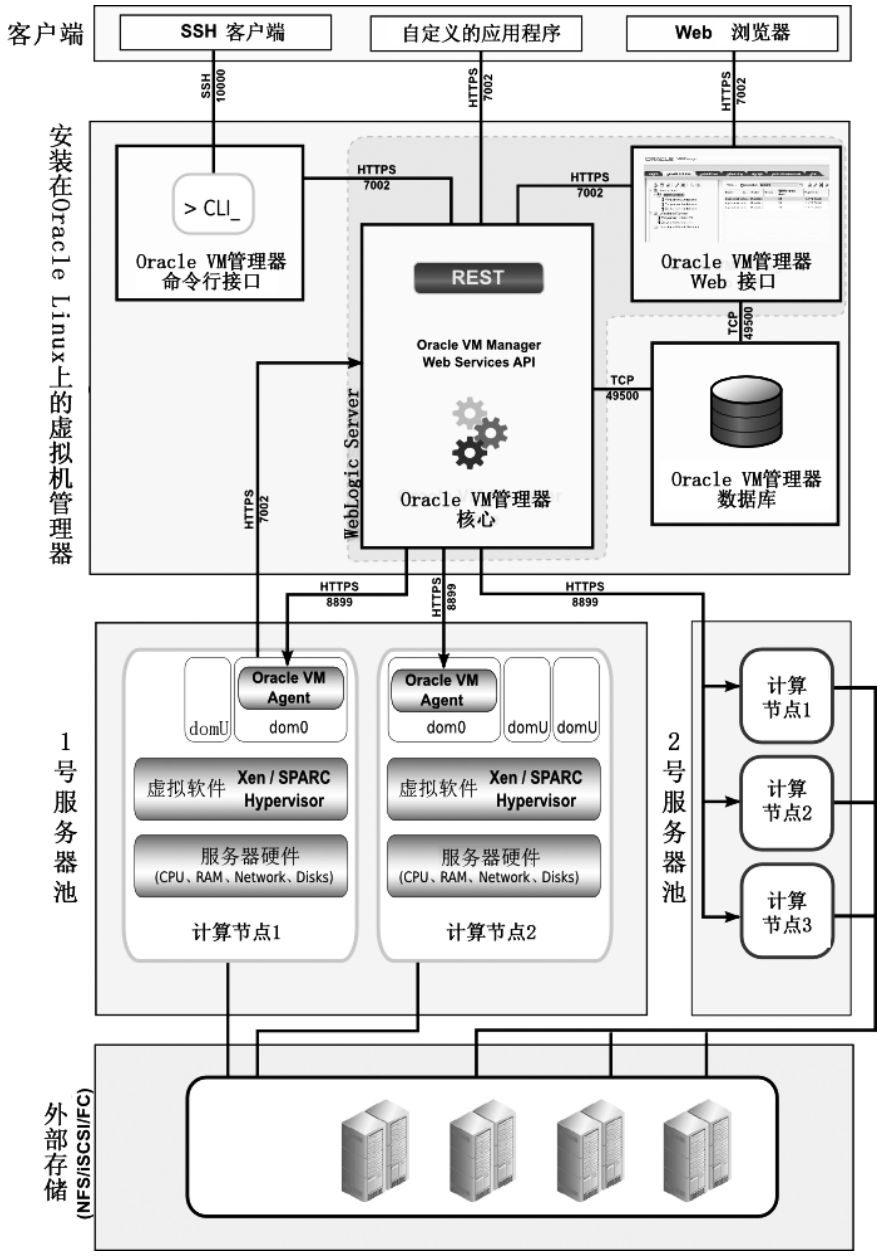


图 4-27　Oracle VM Manager 虚拟环境架构

客户端“自定义的应用程序”是指 Oracle Enterprise Manager 或者采用 Web Service API 编写的应用程序或脚本。

基于 Oracle VM Manager 的云计算解决方案中，支持虚拟机热迁移——在不关闭虚拟机的情况下，从一台服务器移动到同一个服务器池中的另一台服务器上继续运行。在迁移的过程中，虚拟机的状态为 Migrating（迁移）。

4.1.8 应用软件

开源的应用软件数不胜数，开源软件仓库 https://sourceforge.net/和 http://apache.org/index.html#projects-list 保存了大部分的开源软件。下面是一些跟云计算有关的比较出名的软件。

- 用于安全远程登录的软件 OpenSSH，在云端，常用它来建立服务器之间的信任关系。
- 防火墙软件 Netfilter/Iptables，在云端，常用它来建立服务器上的安全屏障。
- 入侵检查软件 Snort，如果防火墙是门卫，那么 Snort 就是巡逻队，配合蜜罐，Snort 效果更佳。
- 开源邮件系统 Postfix，配合其他一些反垃圾和病毒软件，可以搭建一个非常出色的邮件系统，快速部署脚本 iRedMail 能在几分钟内自动部署一个全功能的企业级邮件系统。
- 代理和反向代理软件 Squid，可同时用来执行用户上网行为管理。
- Zabbix 是一个基于 Web 界面提供分布式系统监视及网络监视功能的企业级开源解决方案，它能监视各种网络参数，保证服务器系统的安全运营，并提供灵活的通知机制，让系统管理员快速定位和解决存在的各种问题。
- Odoo 最初是一个 ERP 软件，现在几乎涵盖了企业的所有应用，具备数千个功能模块。采用它开发应用软件就像搭积木一样，直接安装需要的模块即可。

4.2 商业组件

本章开头部分讲到，一套完整的云计算产品包含虚拟化平台、管理工具和交付三部分。注意，这里讨论的是构建云计算的组件，而不是云计算服务，云计算服务提供商需要采用云计算组件来构建云端。目前商业化的云计算组件不多，但云计算服务商却很多。本节对当下比较有影响的商业化云计算组件作简单的介绍，不过这些公司发布了很多组件，却很少讲明这些组件在整个云计算解决方案中处于什么地位、完成什么任务，所以要搞清楚一家企业的云计算组件并非易事。通信协议除 HTTP/HTTPS 外，各个公司都发布了自己的协议。

4.2.1 VMware

VMware 由虚拟机起家，后来不断发展和完善云计算的管理平台和交付部分，如今已经推出了全套的云计算产品。

1．虚拟化平台

VMware 公司的虚拟化技术做得最早，目前应用得也最广，虚拟化产品线丰富，覆盖计算机虚拟化、网络虚拟化、存储虚拟化。其中，计算机虚拟化产品包括桌面版（X86 平台和苹果电脑）和服务器版。计算机虚拟化是云计算产品的核心，桌面版有 Workstation、Fusion（苹果电脑），简化的桌面版 VMware Player Plus 对个人用户是免费的；服务器版有 EXSi，免费版 vSphere Hypervisor 是在 EXSi 的基础上简化而成的，可以直接安装在裸机上。

2．管理工具

VMware 公司的云计算管理平台是 VMware vCenter 系列套件，其中 vCenter Server 是管理控制中心。VMware vCenter 管理私有云，而 VMware vCloud 套件用来管理公有云。

3．交付部分

VMware 的远程桌面协议 PCoIP 是交付部分的核心，此协议基于 UDP，即使在低带宽网络环境下，也会有不俗的表现。在较高速的网络中，能传递高质量的屏幕画面。VMware 公司的客户端能同时兼容微软的 RDP 协议，客户端产品是 VMware Horizon Client。Horizon View Connection 是局域网内的交付网关，Horizon View Security 是因特网的入口网关，Horizon Client 只有通过这些网关，才能访问虚拟化平台上的桌面。

VMware 组合 EXSi、PCoIP、vCenter、View Client 核心组件及其他一些周边工具，推出一系列面向客户应用的解决方案套件，以便降低部署难度。这些套件包括公有云套件 VMware vCloud Suite、办公桌面虚拟化套件 Workspace Suite、混合云套件 VMware vRealize™ Suite、快速混合云部署套件 VMware vCloud® Air™。

采用 VMware 云计算产品搭建的云计算中心架构复杂。

4.2.2 Citrix

Citrix（思杰）公司由远程交付 IT 资源的产品起家，后逐步发展云计算的其他产品线，如虚拟机、管理平台等。不过在竞争激烈的云端产品市场，Citrix 逐步迎难而退，2016 年年初把 CloudPlatform 出售给 Accelerite 公司，自己又重回 OpenStack 社区。Citrix 集中精力发展自己的优势产品——云服务交付，包括软件虚拟化和桌面虚拟化。桌面虚拟化架构如图 4-28 所示。

Delivery Controller（交付控制器）:至少一个。控制器定义了一系列与虚拟化软件平台通信的服务，这些服务完成的任务是：分发应用程序和桌面、验证和管理用户访问、代理用户和其虚拟桌面及应用程序之间的连接、优化使用连接、负载平衡等。

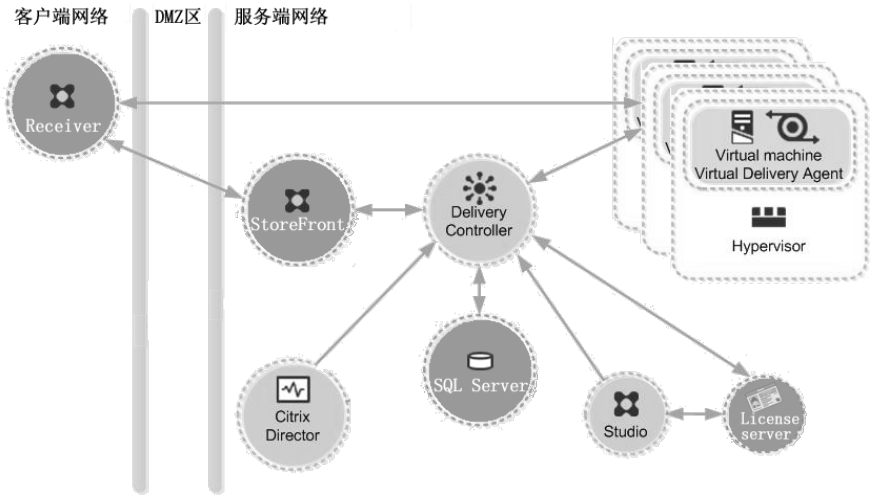


图 4-28　桌面虚拟化架构

StoreFront（存储前端）：StoreFront 验证用户身份，管理用户访问的桌面和应用程序，存储企业的应用程序库，以便让用户自助访问。它还跟踪用户的应用程序订阅、快捷方式名称和其他数据，以确保用户在多个设备上拥有一致的体验。

Director：专门提供给 IT 运维人员和前台服务团队使用的 Web 控制台，通过它可以集中监控 IT 环境、处理问题，以及为终端用户提供支持。

Studio：统一的管理控制台。

1．虚拟化平台

XenServer 早期是由 Citrix 收购 Xen 后推出的虚拟机软件，开源，直接安装在裸机上，核心代码是 Linux 和 Xen。开源是其最大的亮点。

2．管理工具

Citrix 官网上已经没有云计算管理工具的产品了。

3．交付部分

云服务交付是 Citrix 的强项，其交付产品已存在十几年，应该说相当成熟

和稳定了。交付中的通信协议有 ICA 和 HDX，属于 TCP 类型，HDX 是颇具优势的协议，在低带宽网络环境下表现也不俗。交付中的客户端软件 Receiver 是免费的，而且能在目前绝大部分流行的硬件和操作系统上安装运行，如 X86、ARM 等硬件平台，Windows、Linux、Android、Chrome OS 等软件平台。

Citrix 把 XenServer、CloudPlatform、Receiver 打包并加上一些周边产品，推出一些面向应用的方案套件，如统一移动、应用和桌面交付套件 Citrix Workspace Suite、私有办公云套件 XenDesktop 等。其他交付部件还有交付控制器（Delivery Controller）、StoreFront。

相对于 VMware，Citrix 云计算产品成本相对低廉，这是其优势。另外，在企业协同工作方面，Citrix 也有一些不俗的产品，如在线会议系统 GoToMeeting、企业级在线协作应用 Podio、在线培训应用 GoToTraining、DIY 网络研讨会 GoToWebinar。

4.2.3 Microsoft

Microsoft（微软）远程桌面服务架构，如图 4-29 所示。

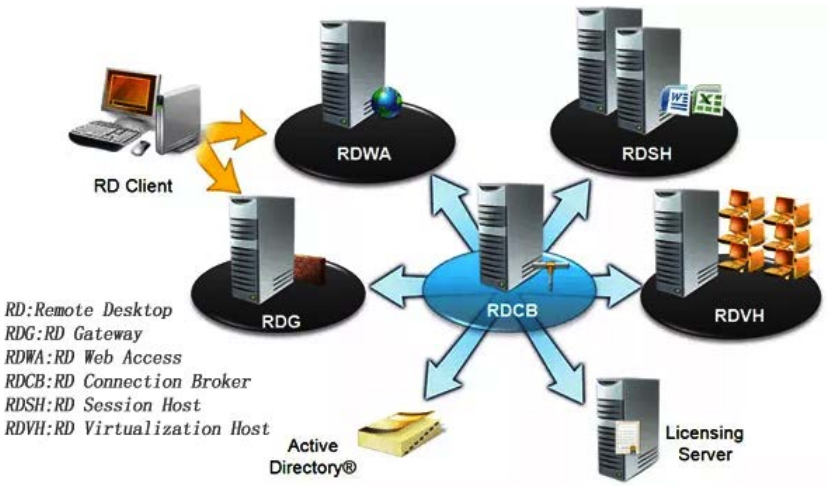


图 4-29　微软远程桌面服务架构

1．虚拟化平台

微软发展虚拟化软件比较晚，刚开始只是一个单机版 Virtual PC，虚拟机里只能安装 Windows 操作系统，后来看到虚拟化是大势所趋，其才在 Windows Server 2008 及以后的服务器操作系统中集成虚拟机 Hyper-V。Hyper-V 作为一个角色由用户选择是否安装，而且管理单机的虚拟机极其简单方便，在 Windows Server 2012 版中，虚拟机软件已经相当稳定了，而且虚拟机支持安装更多类型的操作系统，包括 Linux 操作系统。再后来，微软干脆推出了裸机版虚拟机 Microsoft Hyper-V Server 2012 R2 和 Microsoft Hyper-V Server 2016 R2。微软的远程桌面会话主机（RDSH）是一大特色，这相当于多用户系统桌面，允许多个用户登录 Windows 系统，但是每个用户有自己的桌面，相对于虚拟机桌面来说，RDS 桌面消耗的计算资源更少，同一台计算机允许接入更多的用户。最近一段时间，微软推出了两款容器产品：Windows Server Container 和 Hyper-V Container，后者综合了虚拟机的隔离效果和容器的轻量特征。

2．管理平台

System Center 2016 是微软推出的最新的云计算管理平台。

3．交付部分

RDP 通信协议起源于思杰的 ICA，后来由微软独立发展，由此推出 RDP 7 版、RemoteFX 和 RDP 8 版，在用户体验方面每个版本都有大幅度提高，在局域网内具有不错的表现。客户端软件 mstsc.exe 一直随 Windows 操作系统自带，最近微软也发布能在 Android、Linux、Mac OS 上运行的客户端，但是用户体验远远不及 Windows 版。远程桌面代理（RD Connection Broker）对用户会话进行负载均衡，远程桌面网关（RD Gateway）和远程桌面 Web 网关是交付入口点，但不是必需的。

微软的产品线极其丰富，几乎覆盖了全部的软件层面，从最底层的操作系统到最上层的应用软件，应有尽有。如果采用微软的产品搭建云计算环境，则架构最为简洁。

第 5 章

“云”技术

我们已经知道，一个完整的云系统包含 5 个角色，其中云服务提供商和云服务消费者是最主要的角色，这两个角色缺少任意一个都不能成为“云”。由这两者组成的“云”，算是最基本的“云”，如本地私有云就是只有供需双方的基本云。本章开始重点介绍组建云端的技术、云端与终端的通信协议，最后顺带简单介绍一下云终端设备。云端、通信协议（加上网络）和云终端简称为云计算的“云”、“管”、“端”。在建设具体的云端时到底采用什么技术，这与终端用户的类型密切相关，终端用户的类型间接决定了云服务的模式，以及需要在云端运行的软件种类。总之，要根据云服务消费者的需求来选择技术并组建相应的云端，需求不同，组建云端的技术也不尽相同。

5.1 云端技术

云端是云计算的核心，必须紧贴用户需求，同时应具备水平伸缩性——

在需要的时候可随时增加服务器，从而扩充云端计算能力；在不需要的时候可自动让多余的机器睡眠，以达到节能减排的目的。

前面讲过，软件是让 CPU 来完成某项任务的步骤，云端是软件运行的场所，因此可以这样说：云端是运行各种软件来完成相应任务的地方。云端采用的技术与任务的大小有关，大型任务（如核爆模拟、天气预报）和小型任务（如四则运算计算器、文字编辑等）采用的技术明显不同，前者很难用一台物理计算机按时完成任务，所以需要联合多台计算机（称为集群）共同完成任务；而后者占用一台物理计算机就浪费了资源，所以需要对单台物理计算机进行分割（称为虚拟化或容器化）。将单台物理计算机分割成多台虚拟机，每台虚拟机完成一个任务，这意味着一台物理计算机并行执行多个任务。如果一个任务需要由一台物理计算机来完成，那么就直接分配一台物理计算机给此任务，这样的任务就是中型任务。如图 5-1 所示可以很好地表示这种关系。

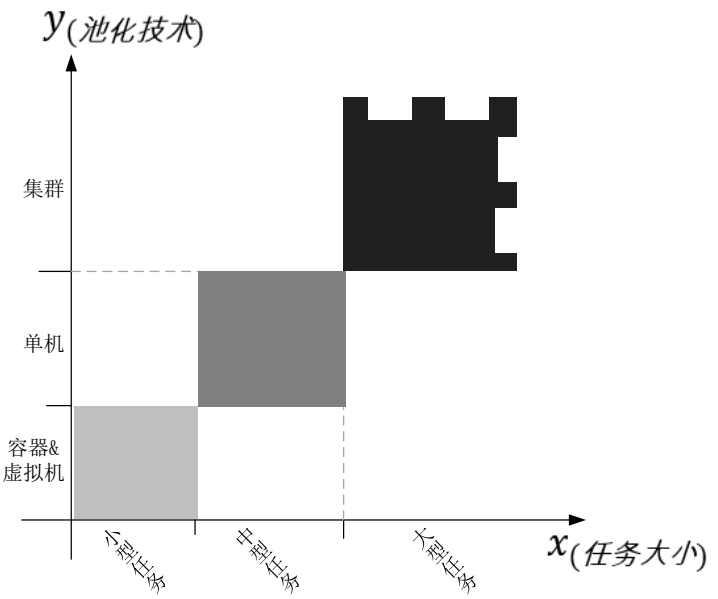


图 5-1　技术与任务的关系

对一台物理计算机做分割（虚拟出很多更小的计算机），然后给小型任务分配一台虚拟机，从而充分利用资源。虚拟机的大小是由 CPU 速度、内存大

小、硬盘容量和每月的网络带宽定义的，当然虚拟机越大，租金就越高。由一台物理计算机无法完成的大型任务，就要采用多台物理计算机集群在一起组成分布式系统。分布式系统还涉及任务划分与分配、多机协同工作的问题，后面会做进一步阐述。

下面讲述的各种云端技术不一定在所有云端都会采用，一些小型云端（如家庭云、小企业办公云等）根本不需要这么复杂的技术。

5.1.1 云布局

前面已经讲到，延时是网络的一个关键属性，那么对于实时的强交互性软件来说，云端至终端的往返延时应该控制在 100 毫秒之内，否则就会大大降低操作云端软件的用户的体验。比如编辑一份 Word 文档，输入的文字要过一会儿才能出现在屏幕上，这是谁都接受不了的。目前云化实时强交互性软件的途径有两条：一是采用网页浏览器；二是在人口密集区建立云端分部，从而降低网络延时。

降低延时的方法就是减少终端与云端的网络路径上的转发节点数目或者提高网络设备的转发速度，但我们不能直接修改广域网的网络，唯一能做的事情就是尽量把云端建在离用户最近的地方，这就是“让计算离用户最近”的原则。这里的“近”不是指地理位置上的近，而是指网络延时小。比如美国的国际接入中心与中国的广州，地理位置上相距很远，但是网络延时却非常小，因为连通两地的是太平洋海底光缆，中途转发设备极少，几乎没有延时，所以两地相距很“近”。

在中国，较好的布局是：分别在北京、上海、广州、沈阳、南京、武汉、成都、西安人口密集区部署云端分部。这 8 个城市也是组成电信网的核心网部分，国际出口设在北京、上海和广州，所以如果云端要服务全球，最好的地方就是北京、上海、广州三个城市。

对于一个服务全国的公有云端来说，一个不错的布局方案如下：

（1）在北京、上海和广州三地建设存储云，专门存储租户数据，这三个存储云互为备份，以确保数据万无一失。

（2）在省会城市分别建设一个中等规模的云端分部，这些云端分部的主要作用是计算，而不是保存数据，数据被保存到存储云。

（3）在地级市建设小型云端支部，完成辖区内的计算任务。

（4）如有必要，在县级市建设微型云端。

这些云端是各自独立的，没有从属关系，但是它们都与存储云建立联系。在租户登录时，会自动引导其进入最“近”的云端。比如，租户甲平时都在中山，自然是登录中山的云端，当他出差去北京时，就登录到北京的云端。公有云全国布局如图 5-2 所示。

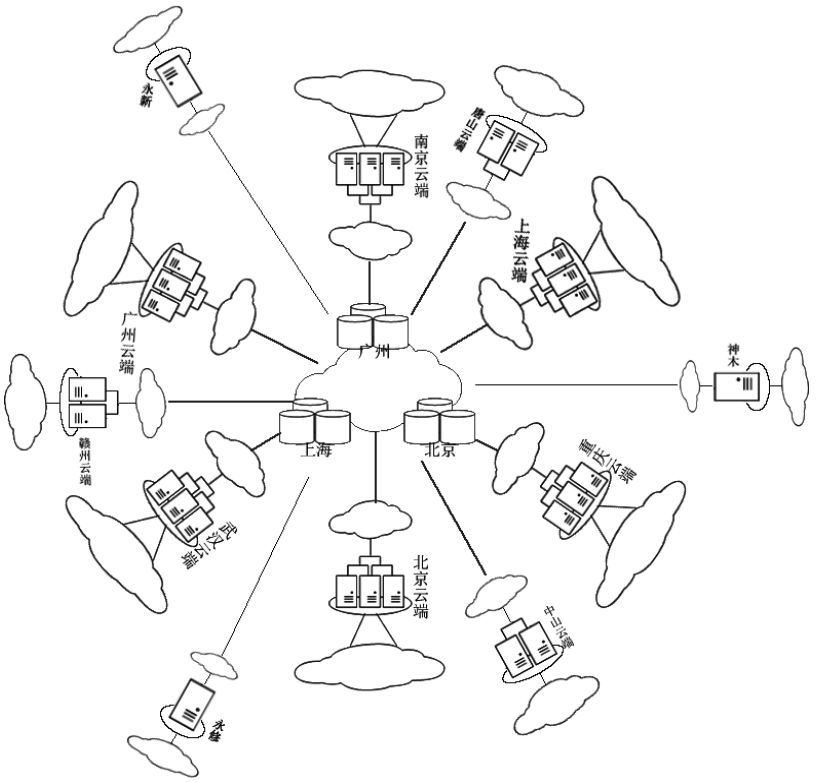


图 5-2　公有云全国布局

这样，各个计算云与存储云之间的数据传输是在后台批量进行的，而计算云与终端之间是前台的、批量的或者实时的。对租户来说，实现了就近计算，从而可以大大降低网络延时。如果有国外租户，则可能还要在国外建立云端。

提醒：云端都应该规划成可伸缩性的——开始运行少量服务器，以后再根据租户数的增加不断增加服务器。

相比于“让计算离用户最近”的原则，目前有一个类似的概念，叫作“内容分发网络”，可以理解为“让内容离用户最近”，全称是 Content Delivery Network，简称为 CDN。用户就近访问网络内容（如网站、流媒体等）有两个好处：一是用户体验好，二是传输成本低。如图 5-3 所示为阿里云 CDN。

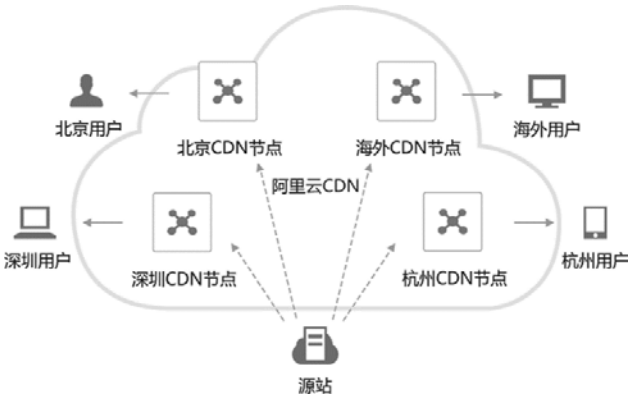


图 5-3　阿里云 CDN

除上面的“让计算离用户最近”和“让内容离用户最近”的布局原则外，还有“离能源丰富的地方最近”和“离寒冷的地方最近”的布局原则。云中心的能耗主要包括机器设备消耗的电力和温湿度控制设备消耗的电力，所以充足稳定的能源供应至关重要，同时尽量降低温湿控制设备的能耗，比如把云中心建在寒冷的地区（如南极洲），就不需要制冷设备了。在传统的机房里，制冷设备的能耗会超过一半。最理想的建设云端的地区的特点有：寒冷、电力充裕、人口密集、无地质灾害、计算机网络设施发达，著名云中心选址如图 5-4 所示。

图 5-4　著名云中心选址

5.1.2　存储

前面讲过，云计算的精髓就是把有形的产品（如网络设备、服务器、存储设备、各种软件等）转化为服务产品，并通过网络让人们远距离在线使用。而计算资源主要是指服务器（CPU、内存）、存储和网络，存储一方面是虚拟内存的组成部分，另一方面也是软件、数据的存放场所，存储如图 5-5 所示。

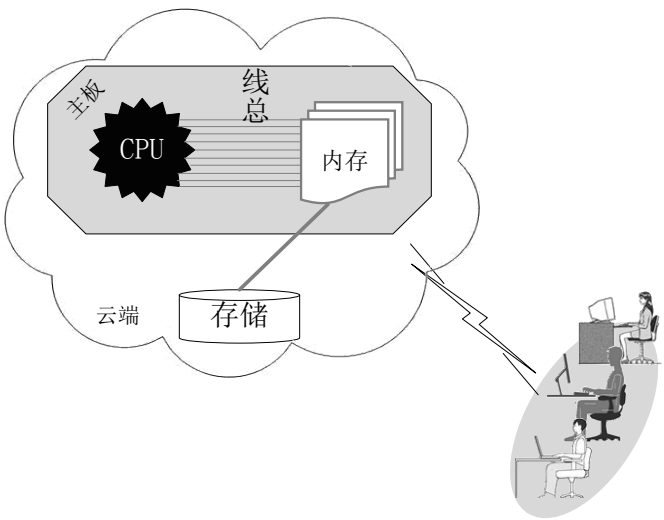


图 5-5　存储

CPU 和内存通过主板紧密地捆绑在一起，二者利用主板上的高速并行总线进行通信，目前的技术还不能做到使它们分离，而且也没有这个必要。但是如果存储与 CPU 分离（不直接通过主板连在一起），就会有很多好处，比如可以共享存储、计算机可以无状态、便于计算资源横向伸缩等。目前能实现共享的技术很多，如 FC、FCoE、iSCSI、NFS、CIFS 等，前三个是磁盘块共享技术，后两个是文件共享技术，而且还出现了块共享的 SAN 产品和文件共享的 NAS 产品。一个磁盘块等于数个整磁盘扇区，一个磁盘扇区能储存 512 个字节的数据（现在大容量硬盘是 4KB 扇区），而扇区是读写硬盘的最小单位。也就是说，一次不能从硬盘读写小于一个扇区的数据（512B 或 4KB）。直接读取磁盘块不需要操作系统参与，但是读取硬盘上的文件（如复制“我的简历.doc”到 U 盘上）必须要操作系统配合。根据存储与 CPU 分离的程度，存储可划分为如下 3 种类型。

（1）**外部存储**：存储和 CPU 不在同一台计算机上，如 SAN 和 NAS 存储是单独的存储设备，它们通过以太网线或者光纤与计算机连接。专门的存储网络设备很贵，随着以太网速度越来越快，基于以太网的存储技术逐渐流行起来，如 iSCSI，10Gbit/s 的网卡能提供 1GB/s 的理论速度。注意，这里的单位 Gbit/s 和 GB/s，前者表示每秒多少比特，一个比特就是一位二进制数字，要么是 0，要么是 1；后者表示每秒多少字节，一个字节等于 8 个比特。计算机里的一个字节先要加上一位校验位和一位停止位，然后再通过网卡传递出去，所以一个字节传递到网络上就占据了 10 个比特位。外部存储如图 5-6 所示。

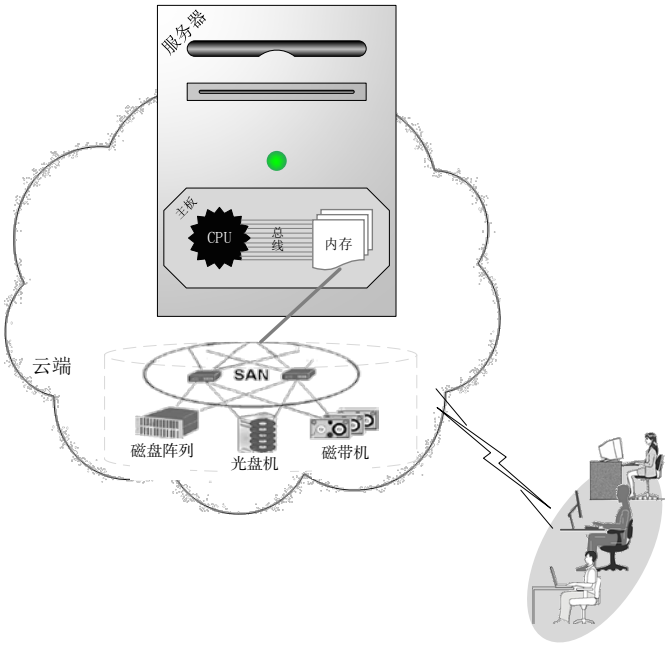


图 5-6　外部存储

（2）**直接存储**：存储模块直接接插到主板上，通过 PATA、SATA、mSATA、SAS、SCSI 或者 PCI-E 接口总线通信。传统的机械硬盘一般采用 PATA、SATA、SAS、SCSI 接口，相对于外部存储，直接接插主板的机械硬盘的速度优势越来越不明显，但是固态硬盘（如 mSATA、PCI-E）的速度优势还是比较明显的，尤其是 PCI-E 的固态硬盘，代表着业界顶尖的存储技术。直接存储如图 5-7 所示。

（3）**分布式存储**：通过分布式文件系统把各台计算机上的直接存储整合成一个大的存储，对参与存储的每台计算机来说，既有直接存储部分，也有外部存储部分，所以说分布式存储融合了前面两种存储方案。由于需要采用分布式文件系统来整合分散于各台计算机上的直接存储，使之成为单一的名字空间，所以所涉及的技术、概念和架构非常复杂，而且还要消耗额外的计算资源。服务器存储局域网（Server SAN）逐渐被数据中心采用，而且发展很快，Ceph 分布式存储系统就属于 Server SAN，被很多云中心采用。目前的软件

定义存储（SDS）概念就是分布式存储。分布式存储如图 5-8 所示。

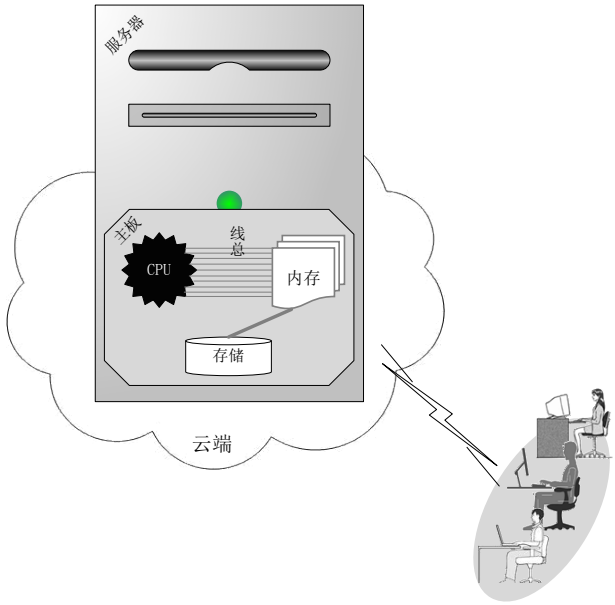


图 5-7　直接存储

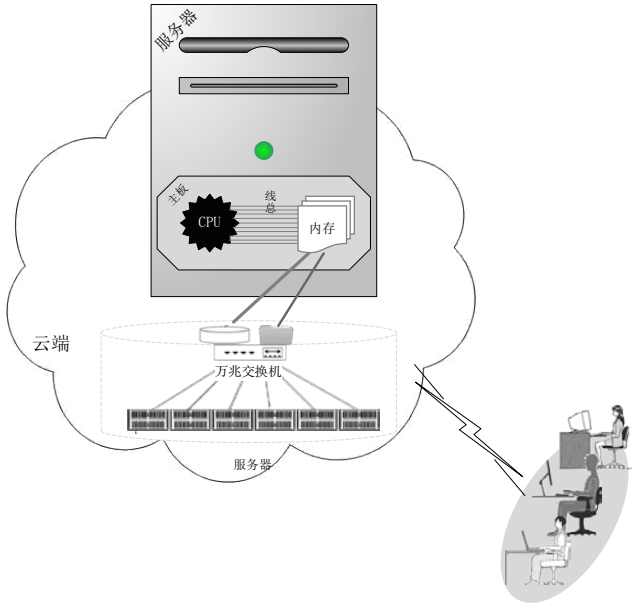


图 5-8　分布式存储

存储的评价指标有容量、速度、每秒读写次数（IOPS）、可用性。“容量”比较容易理解，就是能存储的数据总量。在实际项目中，我们更在乎有效容量，比如四块 1TB 的硬盘，加起来容量是 4TB，但是如果这四块硬盘做成镜像（RAID-1），那么有效容量就是 2TB；如果做成 RAID-5，有效容量就又不一样了。容量要求很容易满足，一般采用横向扩容方案即可。“速度”指标是指每秒传输的数据量，速度与带宽是同一个概念。

IOPS 是最重要的指标，定义为每秒钟能响应的读（或写）操作的次数，体现的是并发性和随机访问能力。IOPS 与磁盘的转速、平均寻道时间密切相关，磁盘的平均寻道时间为 4 ~ 12ms，对于转速 7200RPM 的磁盘，我们可以计算出其 IOPS 的近似值：1000÷[1000÷(7200÷60) ÷2+8]=83。对单块磁盘来说，“读/写”磁盘从微观层面上看是串行的，比如 100 个人同时访问磁盘，磁盘是一个一个地响应用户的请求，但在宏观上又表现为并行性，即 100 个人在一秒钟内同时访问到了磁盘，给人一种并行的错觉。提高 IOPS 的方法有很多，比如采用更好的硬盘（如固态硬盘），或者增加磁盘的数量并让访问分散到各个硬盘，也可以采用更多的缓存（Cache），从而让经常访问的内容驻留在缓存中。

如图 5-9 所示，如何将主机 IOPS 需求转换成磁盘实际 IOPS 负载，并计算出需要的磁盘数呢？

首先我们要明白，主机的 IOPS 需求并不一定等于磁盘实际 IOPS 负载。比如对于 RAID-1，主机写一次，磁盘实际要写两次（镜像的两个磁盘各写一次）；再比如 RAID-5 存储，主机写一次，磁盘要读两次、写两次，共 4 个 IOPS。主机写 1 次对应磁盘实际发生的读写次数称为写惩罚（Write Penalty），阵列类型不同，写惩罚也不同，RAID-5 写操作如图 5-10 所示。

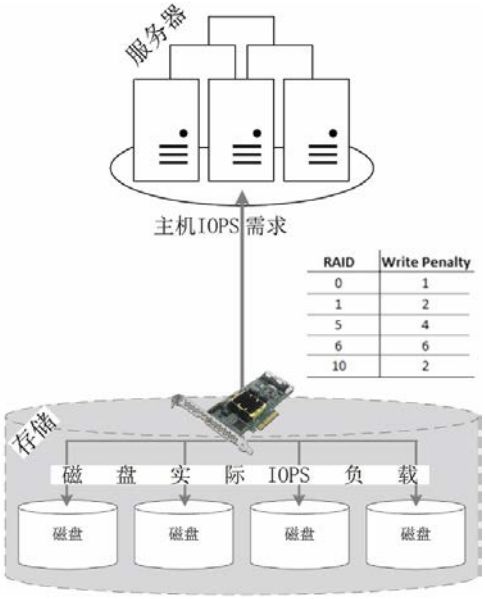


图 5-9　主机 IOPS 需求与磁盘实际 IOPS 负载

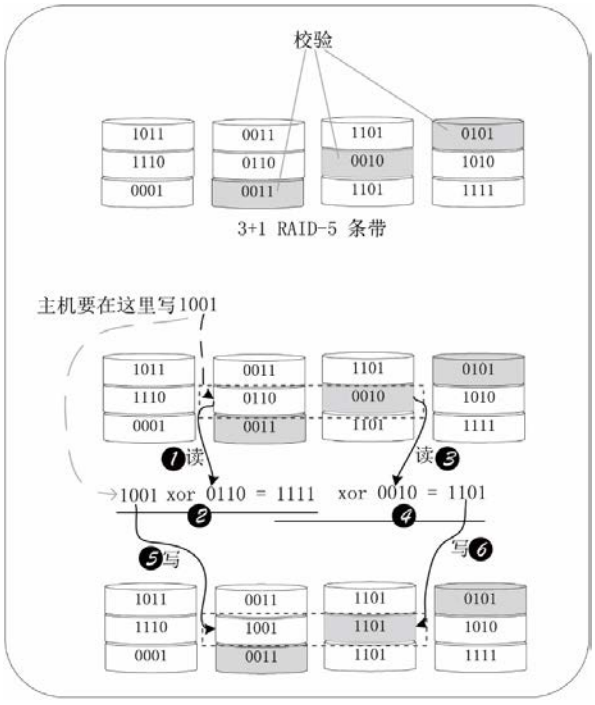


图 5-10　RAID-5 写操作示

在图 5-10 中，主机向 RAID-5 存储写 1001，存储中实际发生的步骤如下。

（1）读取原始数据 0110 并与新的数据 1001 进行异或操作：0110 xor 1001=1111。

（2）读取原有的校验位 0010，并用第一步算出的数值 1111 与原校验位再做一次异或操作：0010 xor 1111=1101。

（3）然后将新数据 1001 写入数据磁盘，将第二步中计算出来的新校验位 1101 写入校验磁盘。

现在假设主机的 IOPS 需求是 x，读/写比率为 2∶1，存储为 RAID-5，那么可以算出磁盘实际的 IOPS 负载 y：

$$y = x \times \frac{2}{3} + x \times \frac{1}{3} \times 4$$

例如 x = 360，则计算得出 y = 720。假设一个 7200RPM 的 SATA II 硬盘的 IOPS 按 90 算，那么需要 $\frac{720}{90} = 8$ 个磁盘，8 个磁盘做成 7+1 RAID-5。在上述公式中，“$x \times \frac{2}{3}$”表示每秒磁盘的读次数，“读”不存在惩罚，也就是说，主机的一次读对应磁盘的一次读。

存储的“可用性”指标关乎数据的安全性，是指保存在存储设备上的数据不会丢失的概率。由于磁盘故障而导致用户数据丢失，是一件非常糟糕的事情。通常用几个“9”来衡量可用性，比如 3 个“9”的可用性，意思是一年之中 99.9%的时间内数据不会丢失。提高可用性的方法也有很多，比如购买更稳定可靠的磁盘、做成多路镜像、采用不间断电源供电、提高备份的频率，甚至是建立多个异地灾备中心，乃至部署双活中心等。对于当前主流的硬盘产品，我们总结如下，如表 5-1 所示。

表 5-1　当前主流的硬盘产品

	容　量	带　宽	IOPS	可 用 性
5400 转 PATA 硬盘	128GB~2TB	1Gbit/s	30~50	如果损坏，还可以恢复大部分数据
7200 转 SATA 硬盘	250GB~3TB	3Gbit/s	75~100	
10000 转 SATA 硬盘	300~800GB	3 或 6Gbit/s	125~150	
10000 转 SAS 硬盘	100~600GB		140	
15000 转 SAS 硬盘	100~300GB	3~12Gbit/s	175~219	
SATA SSD	8GB~1TB	3Gbit/s	400~20000	如果损坏，则整块硬盘上的数据全部丢失
SATA SSD	8GB~1TB	6Gbit/s	60000~120000	
mSATA SSD	80GB~1TB	4~8Gbit/s	400~20000	
PCI-E SSD	60GB~2TB	>8Gbit/s	12~1000 万	

注意，表 5-1 中的带宽单位，Gbit/s 是每秒多少比特，一字节等于 8 比特。在带宽的计算中，有经验公式：1B≈10bit，加上了校验位和停止位。比如你家里申请了 4 兆的宽带上网，那么在网络通畅的情况下，下载文件的速度约等于每秒 400KB，即 400KB×10=4MB。

在云端，往往设计两个层次的存储：一个是物理机器访问的存储，一个是虚拟机访问的存储。

5.1.3　虚拟化与容器技术

虚拟化是云计算的重要技术，主要用于物理资源的池化，从而可以弹性地分配给用户。物理资源包括服务器、网络和存储。但是计算资源的池化不一定要用虚拟化技术，金属裸机也能池化，比如 IBM 的 SoftLayer 就是直接使用物理机来实现云计算的。

主机虚拟化的思想可以追溯到 IBM 机器的逻辑分区，即把一台 IBM 机器划分成若干台逻辑的服务器，每台逻辑服务器拥有独占的计算资源（CPU、内存、硬盘、网卡），可以单独安装和运行操作系统。IBM 机器价格昂贵，相对于当时的计算任务来说，机器的计算能力太过强大，所以需要划分为更小的计算单元。后来随着个人计算机处理能力的不断提升，1998 年 VMware 公司成立，这家公司专注于机器虚拟化的软件解决方案。也就是说，对于不支

持逻辑分区的计算机，可以直接通过安装 VMware 虚拟化软件来模拟更多的虚拟机，然后再在这些虚拟机里安装操作系统和应用软件，可以给虚拟机灵活配置内存、CPU、硬盘和网卡等资源，虚拟机结构如图 5-11 所示。

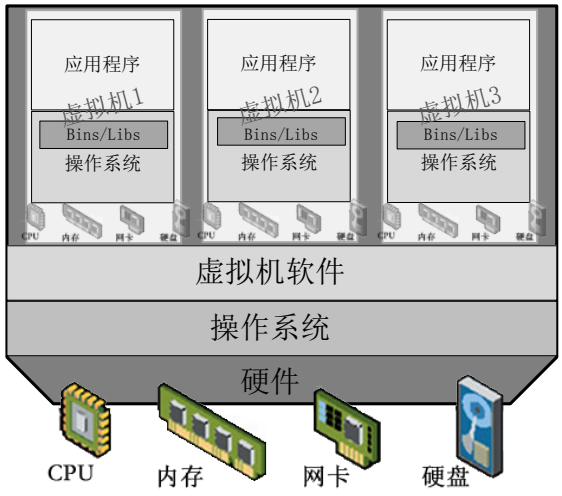


图 5-11　虚拟机结构

在一台物理机上可以创建很多虚拟机，虚拟机里允许安装不同的操作系统，配置不同的网络 IP 地址。

可以动手做个实验：购买一台计算机（配置：双核 3.0GB CPU、4GB 内存、500GB 硬盘、1000MB 网卡），首先安装 Windows 8 操作系统，把所有的硬件驱动安装好，然后安装办公软件、QQ、音视频播放软件和 C 语言开发工具等。接下来安装 VMware Workstation 12.0 虚拟化软件，安装完成后重新启动计算机，并双击桌面上的“VMware Workstation”图标启动虚拟化软件，如图 5-12 所示。

在里面可以创建很多虚拟机，比如图 5-12 中就创建了 14 台虚拟机，虚拟机里安装的操作系统分别是 Windows 7、Windows 8、Windows Server 2012、Windows XP 和 Linux 的各种发行版。只要不启动虚拟机，就不会消耗内存和 CPU 资源，但是会占用硬盘空间。能同时启动的虚拟机数量与计算机的物理内存容量和 CPU 计算速度有关。另一款免费的桌面虚拟机软件是 Oracle 公司的

VirtualBox。

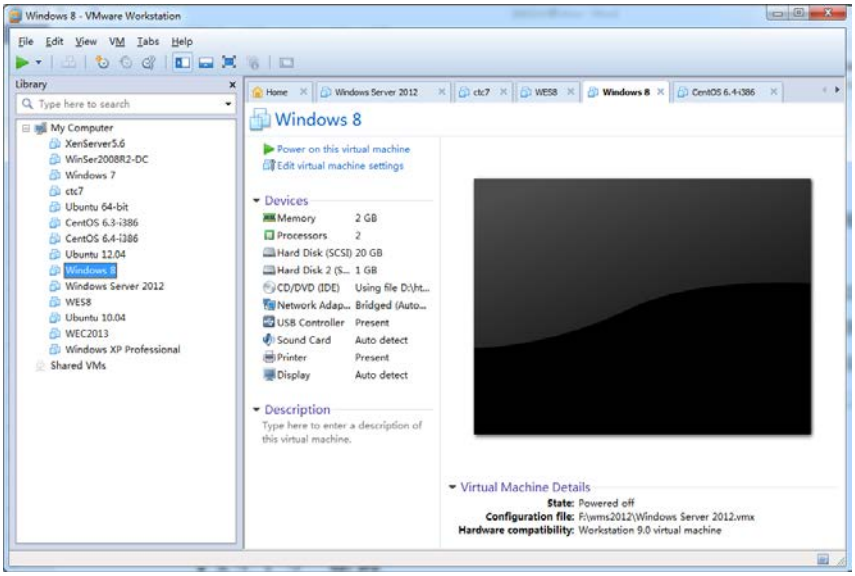


图 5-12　启动虚拟化软件

近几年来，很多大公司不断加入主机虚拟化软件市场，竞争异常激烈。排名靠前的有 EMC（收购 VMware）、微软、思杰、红帽、Oracle、Parallels。微软把虚拟机直接集成在操作系统里；红帽携 KVM 开源虚拟机一路攻城略地；Oracle 的虚拟机算是个小字辈；Parallels 公司的产品既支持虚拟机，也支持容器。容器是一个轻量级的运行环境，在同样配置的物理机上，能同时运行比虚拟机多三倍的容器。VMware 的虚拟化产品受到微软 Hyper-V、红帽 KVM 及思杰 Xen 的强大冲击，其市场开始出现萎缩的征兆。

目前 CPU 发展到多核，且本身就支持虚拟化。虚拟化软件厂商直接推出了能运行在裸机上的虚拟化软件层，如微软的 Windows Hyper-v 2012、EMC 的 ESXi 6、思杰的 XenServer、红帽的 RHEV-H 等，然后在虚拟化软件层上直接创建更多的虚拟机，如图 5-13 所示。

虚拟化软件层消耗的计算资源很少，一般在 10%以内，相比前面的方法，同一台物理机可以运行更多的虚拟机。针对云计算方案，各虚拟化软件厂商

还推出了云端虚拟机管理工具，实现虚拟机的创建、删除、复制、备份、恢复、热迁移和监控等统一管理。其中，热迁移就是在不关闭虚拟机的情况下，把虚拟机从一台物理机转移到另一台物理机上，而正在使用虚拟机的租户感觉不到虚拟机被移动了。一台虚拟机的计算能力目前还不能超过其所在的物理机的计算能力。

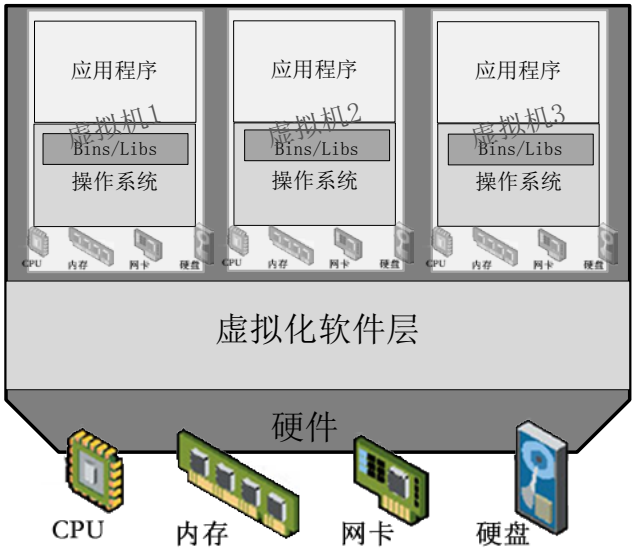


图 5-13　直接在虚拟化软件层创建更多虚拟机

但是在每台虚拟机里都要安装和运行操作系统的做法，仍然浪费了很多计算资源。举一个简单的例子：假如一台计算机的配置是双核 3GHz 的 CPU、8GB 的内存、500GB 的硬盘，现在在这台计算机上创建 6 台虚拟机，每台虚拟机分配 1GB 内存、64GB 硬盘、1GHz CPU，虚拟机都安装 Windows 7 操作系统。那么当全部虚拟机启动后，几乎就很难再运行其他应用程序了，因为内存和 CPU 资源都被操作系统本身消耗掉了。为此，有公司专门推出了应用软件容器产品，即在操作系统层上创建容器，这些容器共享下层的操作系统内核和硬件资源，但是每个容器可单独限制 CPU、内存、硬盘和网络带宽容量，并且拥有单独的 IP 地址和操作系统管理员账户，可以关闭和重启。与虚拟机最大的不同是，容器里不用再安装操作系统，因此浪费的计算资源也

就大大减少了，这样同样一台计算机就可以服务于更多的租户，容器架构如图 5-14 所示。

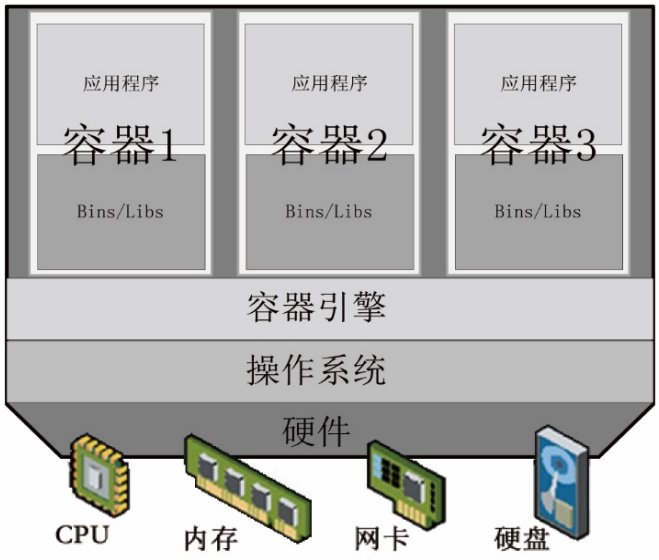


图 5-14　容器架构

但是容器的隔离效果比虚拟机差，在安全性要求高的场合（比如公有云）中，虚拟机使用率要远远高于容器，而安全性要求不高的场合（比如私有云、公司内部 IT 环境）中，容器的使用率会越来越高。谷歌云目前的做法是在虚拟机里跑容器，并把 Kubernetes 交给租户使用。开源产品 Kata 和 gVisor，以及微软的闭源产品 Hyper-V Container 都是跨界产品——既接近于容器的轻量性，又接近于虚拟机的安全性。gVisor 是谷歌主推的开源项目，具备良好的发展势头。

现在有企业推出了云计算一体机（或称为箱式数据中心），做一个大箱子，里面封装了几台服务器、磁盘柜、计算机网络设备、UPS（不间断电源）、制冷设备等，并且安装了虚拟化软件和云管理工具。微软的云计算一体机采用水冷系统，冷水进去，热水出来，如图 5-15 所示。

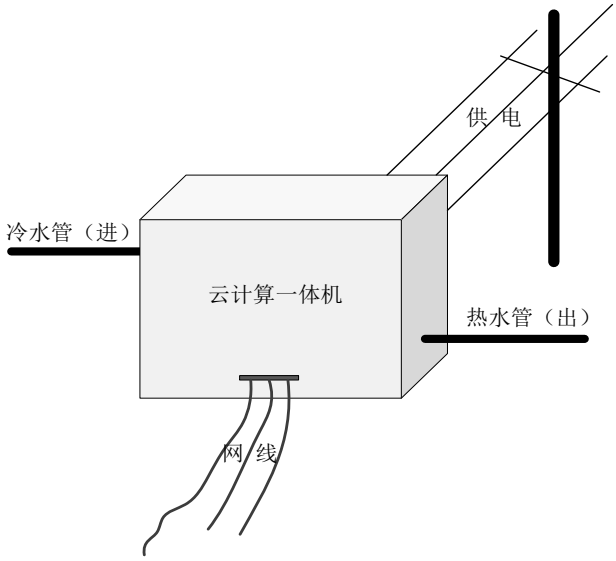


图 5-15　微软云计算一体机

云计算一体机消耗的电能（3000W 以上）全部转化为热能被水带出，热水可以进一步利用，真正做到了节约能源，从而使云计算成为名副其实的绿色 IT。图 5-16 所示为华为的 FusionCube 一体机，不过里面没有制冷设备和 UPS，外表显得“傻大黑粗”。

图 5-16　华为的 FusionCube 一体机

电力使用效率（Power Usage Effectiveness，PUE）是目前国际上通行的用

于衡量云端和云计算一体机能源效率的指标，它等于总能耗与 IT 设备能耗的比值。比值越小，表明能源利用率越高，绿色程度越高。目前国外先进的数据中心机房 PUE 值通常小于 2，而我国的大多数数据中心的 PUE 值为 2～3。在云端，IT 设备主要指服务器、存储设备、网络设备，其他的诸如制冷设备、UPS、安防设备等属于非 IT 设备，其中制冷设备功耗几乎占到总功耗的一半。

出租虚拟机属于 IaaS 云服务，IaaS 的另一种产品是出租裸机，即直接把硬件服务器出租，通过服务器上的远程管理卡可以把配置、安装操作系统、开关机等功能整合到租户自助网站上。

最后，对主机虚拟化技术进行总结，如图 5-17 所示。

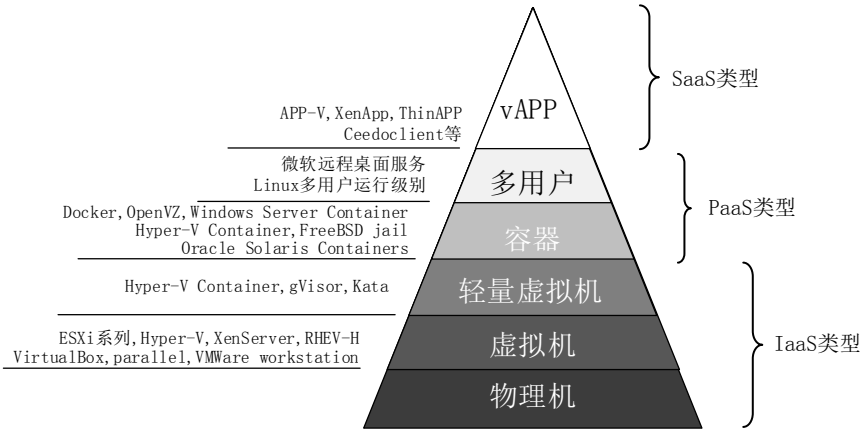


图 5-17　主机虚拟化技术

在图 5-17 中，“多用户”是实现远程桌面最轻量化的方案，即在操作系统里创建多个用户账号，然后让这些用户登录使用计算机。目前微软的远程桌面服务和 Linux 多用户运行级别都是实现“多用户”的方法。最近有不少公司在多用户的基础上进一步做轻量化处理，推出应用程序虚拟化（vAPP），如微软的 APP-V、思杰的 XenAPP、VMware 公司的 ThinAPP、Ceedo 公司的 CeedoClient 等。应用程序虚拟化有一个好处，即使计算服务器按应用分工，比如使用一台服务器专门运行 Word 软件，第二台专门运行 Excel 软件，第三台专门运行 QQ，第四台运行 Photoshop，通过应用软件虚拟化技术把 Word、

Excel、QQ、Photoshop 整合到用户的桌面上。使用户桌面环境与应用程序分离有如下几个好处：

（1）可以根据应用程序来专门定制服务器硬件。不同的应用软件，对机器配置的侧重点不同，比如图形处理软件对显卡要求较高、QQ 对网络要求较高，根据应用软件定制的服务器运行效率自然会提高不少。

（2）节约更多的资源。一台服务器只运行一个应用软件，但同时服务多个用户，可以节约硬盘资源、节约内存资源、节约 CPU 资源，具体原因可以参考操作系统原理和计算机体系结构方面的书籍。

（3）提高生产效率。运行同一个程序，等于执行一份相同的软件代码，从而大大提高了各级缓存的命中率，也加快了进程的切换速度，提高了虚拟内存的换入、换出效率。

图 5-17 中的物理机（也称为裸金属机）是最重型的方案。同一台计算机上租户数越多，表明相应方案越轻量化，反之则越重型化。在实际方案中，要根据租户的需求来确定轻重级别。

目前一台虚拟机不能跨越多台物理机（只能在一台物理机上运行），这意味着虚拟机的运算能力不会超过一台物理机的运算能力，以后的技术能不能突破还很难下定论。目前通过集群联合多台物理机，对外呈现一致的寻址空间，对用户来说，似乎在使用一台超级计算机，但这是虚幻的，集群与虚拟化具有本质的不同。

但是目前主流的主机虚拟化技术都支持过度分配资源，即分配给同一台物理机上的虚拟机的资源之和大于物理机本身的资源数。比如物理机的计算资源是内存 4GB、CPU 8 核、磁盘 100GB，在这台物理机上创建 5 台虚拟机，每台虚拟机分配资源如下：内存 1GB、CPU 2 核、磁盘 30GB，显然 5 台虚拟机资源之和是内存 5GB、CPU 10 核、磁盘 150GB，超过了物理机的资源总数。在虚拟机运行时，按其实际消耗的资源动态分配，但是不超过管理员给其分配的上限，一般计算机正常运行时资源耗费不会超过其总资源的 75%，这样

过度分配资源就容易理解了。CPU 的过度分配率一般为 16 倍，内存的过度分配率一般为 1.5 倍。

接下来我们再来看看网络虚拟化技术。网络虚拟化技术主要用来对物理网络资源进行抽象并池化，以便于分割或合并资源来满足共享的目的。人们很早就意识到了网络服务与硬件解耦的必要性，先后诞生了许多过渡的技术，其中最重要的 6 种分别是虚拟局域网络（VLAN）、虚拟专用网络（VPN）、主动可编程网络（APN）、叠加网络（Overlay Network）、软件定义网络（SDN）和网络功能虚拟化（NFV），网络虚拟化的发展过程如图 5-18 所示。

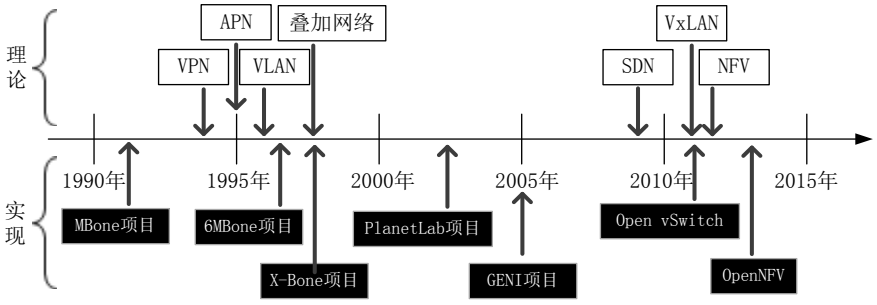


图 5-18　网络虚拟化的发展历程

APN 把控制信息封装到报文内部，路由器根据报文内的控制信息做决策。SDN 和 NFV 是目前最为热门的网络虚拟化技术，在云计算和大数据时代，其发展不可小视。

网络虚拟化技术已经出现了 20 多年，但是发展却一直不温不火，原因是缺少一个杀手级的应用。云计算的出现对于网络虚拟化来说是一次千载难逢的机会，可以说，有了云计算，网络虚拟化才变得如此热门，如果没有网络虚拟化，就没有大规模的云计算。

众所周知，一个计算机网络必须完成两件事：一是把数据从 A 点传送到 B 点；二是控制如何传送。前者主要包括接收、存储和转发数据，后者主要是各种路由控制协议。这跟交通网络很相似，连接两个城市的交通网络具备

的第一个功能就是汽车从一个城市到达另一个城市；第二个功能是控制到底走哪条线路最好。前者就是由公路组成的交通网络，后者就是交通控制系统。

下面我们再来看看传统的网络设备（比如一台路由器）的逻辑分层结构，如图 5-19 所示。

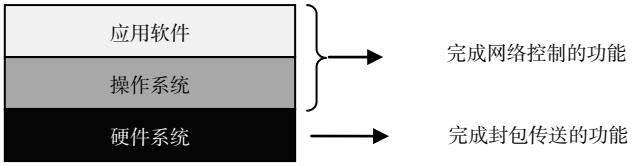


图 5-19　路由器的逻辑分层

传统的网络设备包含了完整的三层，由厂商统一捆绑销售，第三方很难修改里面的软硬件结构，由此对用户来说有几个明显的缺点：一是很容易被网络设备厂商绑定；二是不能快捷满足业务的需求；三是成本高；四是无法实现网络虚拟化；五是数据传送的路径很难保证全局最优。

如今，SDN 是网络虚拟化技术当中最热门的。SDN 技术通过分离网络控制部分和封包传送部分来避免传统网络设备的缺点，处于数据通路上的网络设备蜕化为准硬件设备，网络中的所有网络设备的网络控制部分独立出来由一台服务器单独承担，SDN 原理如图 5-20 所示。

把网络控制部分从各个网络设备中独立出来，统一由 SDN 网络控制器承担，这样做的最大好处是数据传送的路径是全局最优的。SDN 网络控制器类似于 GPS 导航卫星，它存储了全局的网络拓扑图，俯视着整张网络，精确导航着每个数据包的流向。当某台网络设备收到一个数据包时，就会询问网络控制器：“这个包要往哪个口送出去？”SDN 网络控制器可能回答：“从 2 端口送出去。”为了加快转发速度，SDN 网络设备会存储答案，即属于同一个会话的数据包直接从之前的出口送走，类似于现实生活中完成同一个运输任务的车队，在每个交叉路口，GPS 卫星只导航第一辆车，后面的车跟着前面的车行驶即可。SDN 网络设备和网络控制器之间采用 OpenFlow 协议进行通信。

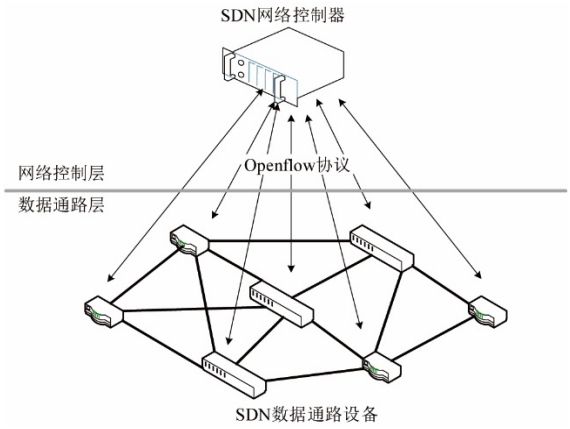


图 5-20　SDN 原理

云端一般采用 Open vSwitch 交换机，它是一款开源的网络虚拟化产品，是二层交换机，性能可以与硬件交换机媲美。利用它可以在虚拟机的下面构筑虚拟网络层，通过实时修改 Open vSwitch 的配置，可以组建变化灵活的局域网，使得一台虚拟机能快速地从一个局域网迁移到另一个局域网中，这是物理交换机所无法实现的，迁移过程如图 5-21 所示。

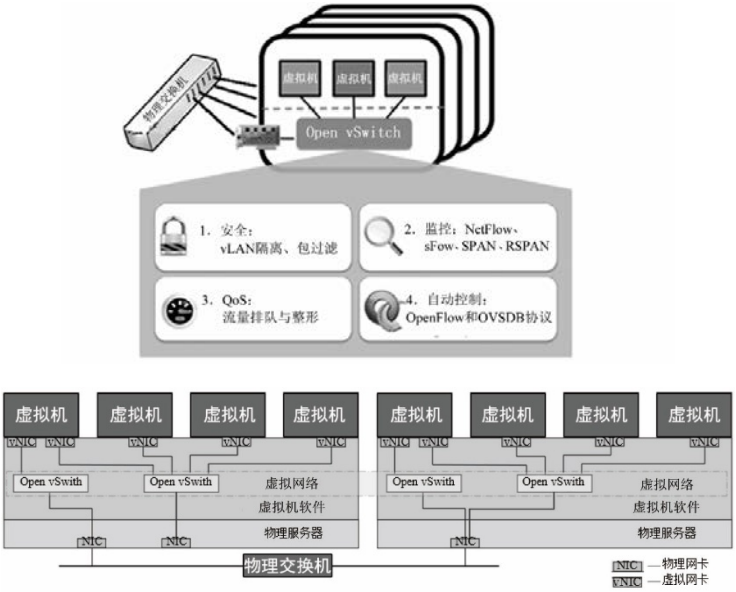


图 5-21　迁移过程

不同物理机上的两台虚拟机之间的网络通路如图 5-22 所示。

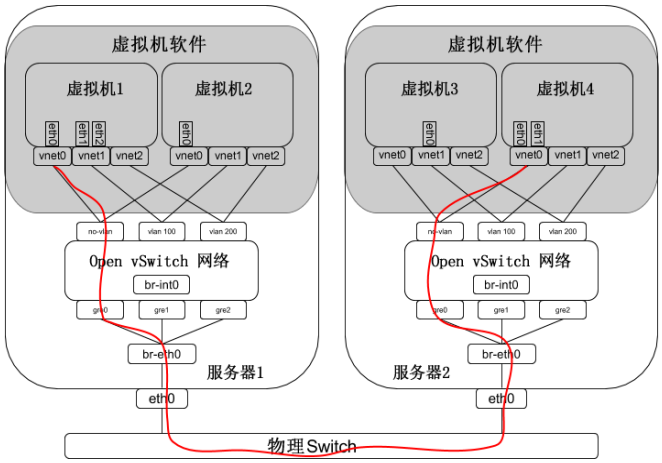


图 5-22　不同物理机上的两台虚拟机之间的网络通路

5.1.4　远程桌面

使用过 Windows 操作系统的用户都知道，计算机启动完毕后看到的那个屏幕叫桌面，上面有很多小图标，包括我的电脑、垃圾桶，以及很多由用户自己创建的应用程序快捷方式。图 5-23 所示是笔者的 Windows 7 桌面。

图 5-23　Windows 7 桌面

图 5-24 所示是安装了 Linux 操作系统的桌面。

图 5-24　Linux 操作系统桌面

不同的用户登录计算机后看到的桌面是不一样的，每个用户都可以定制自己的桌面，而不会影响到同一台计算机上的其他人的桌面。如果把桌面放在云端，就成了远程桌面，租户输入账号和密码就可以登录到云端的桌面，此后使用云端桌面与本地桌面一样，没有区别。有人喜欢把远程桌面称为虚拟桌面接口（VDI），这样学术性更浓厚一些。

使用云端桌面能实现移动办公，请看下面的情景描述：

李昭在一家公司做售后，主要工作是解决客户的问题，其活动范围包括家、公司、客户处。他经常出差，具体工作包括写文档、修改软件 Bug、管理问题库工具。李昭家里有台式机，公司有办公计算机，出差时携带笔记本电脑，公司还给他在私有云上创建了一个虚拟机。李昭在虚拟机里安装了 Windows 8 操作系统，然后又安装了 Office 2013 办公软件、QQ、PDF 阅读器、暴风影音、Visual Studio 2012 开发工具、Photoshop CS5、Chrome 网页浏览器、问题管理工具等软件，最后开启了虚拟机的远程桌面。这样，李昭就能在家里、公司和客户处登录到云端桌面了。云端桌面成了李昭唯一的办公

桌面，如图 5-25 所示。他写的文档、修改的软件、设计的图纸、下载的电影和歌曲等都在这里，不管李昭走到哪里，都可以随时随地访问其数据资料。

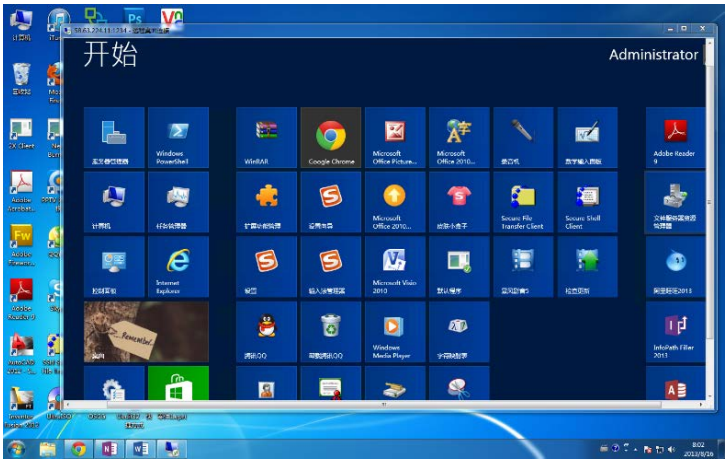


图 5-25　云端桌面

远程桌面协议是实现远程桌面应用的关键技术之一，协议规定终端与云端桌面如何通信。目前的远程桌面协议有微软的 RDP 10、EMC 公司的 EOP/PCoIP、思杰公司的 ICA/HDX、红帽公司的 SPICE，以及 Ericom 公司的远程桌面产品。微软的 RDP 10 客户端软件目前能运行在安装了 Windows 10、Windows Server 2012 的计算机上，或者固化了相应操作系统内核（Windows Embedded Standard 10、Windows 10 IoT）的嵌入式终端上，同时发布了针对安卓和苹果操作系统的客户端软件。而云端桌面的操作系统只能安装 Windows 8、Windows Server 2012 和 Windows Server 2016，比如图 5-25 就是在安装了 Windows 7 操作系统的计算机上登录到云端的 Windows 8 桌面所看到的开始菜单。

开源项目 FreeRDP 支持 RDP 协议，能在很多操作系统上运行，但是目前不支持 RDP 8.0。微软的 RDP 8.0 协议规范公开在官方网站上，预计不久后，FreeRDP 就会支持 RDP 8.0 了。而其他几个公司同时发布了能运行在 Windows、Linux、Andriod 和 Mac OS 系统上的多个版本。

有三种方法实现远程桌面：

第一种是基于 IaaS 云服务的虚拟机或裸机，租户租用云服务提供商的虚拟机或裸机，然后自己安装操作系统、应用软件并开启远程桌面，我们称这种远程桌面为 IaaS 云桌面。

第二种是 IaaS 云服务的应用软件容器，租户共享底层的操作系统内核，单独安装应用软件和一些基础运行库，我们称这种远程桌面为 IaaS 容器桌面。比如 OpenVZ 和 Virtuozzo Containers 实现 Linux 的容器桌面，Windows Server Container、Hyper-V Container 和 Virtuozzo Containers 实现 Windows 的容器桌面。

第三种基于半平台 PaaS 云服务，并为每个租户创建一个系统账户，其实就是利用了现代操作系统的多用户特点，即同时让许多人登录并使用计算机，我们称这种远程桌面为 PaaS 云桌面。

PaaS 云桌面在私有云中应用比较普遍，尤其是私有办公云。因为搭建 PaaS 云桌面具备很好的伸缩性，所以小的云端用一台计算机即可。一台配置高的台式机（4 核 3.0GB CPU、16GB 内存、1T 硬盘）就可以供 20 位左右的员工日常办公，以后随着公司的发展，员工不断增多，可以不断地增加云端计算机来构建更大的云端。目前微软的终端服务技术就是典型的 PaaS 云桌面的解决方案，要求每个用户购买许可证，在正规的商业环境下建议购买正规许可证。

由于 IaaS 云桌面是每个租户单独使用一台云主机（虚拟机或裸机），所以租户间的隔离效果最好，IaaS 容器桌面的隔离效果次之，PaaS 云桌面的隔离效果最差。这里的“隔离”，是指一个租户的操作行为和私有数据资料对另一个租户的可见度。比如两个租户登录到同一台计算机的 PaaS 云桌面，那么他们之间是互相可见的：可以看到对方的进程、对方的数据文件，一方关机另一方就会被强行退出，具体内容可参阅后面的 5.1.9 节。

在规划具体的云端方案时，要综合考虑到底采用什么类型的桌面，这里提供一些经验如下：

（1）同一个部门的人采用 PaaS 云桌面。

（2）中小企业采用 PaaS 云桌面。

（3）重要岗位（如财务、人事、销售）采用 IaaS 云桌面。

（4）同一个研发团队采用 IaaS 容器桌面。

（5）公有云采用非 PaaS 云桌面。

（6）领导和核心保密部门应采用裸机桌面。

5.1.5 负载均衡

众多的租户通过计算机网络接入云端的计算机并使用里面的计算资源，这里有三点需要注意：一是租户很多，二是云端有很多计算机，三是进入云端的宽带专线条数有限（一般为 1～3 条光纤），如图 5-26 所示。

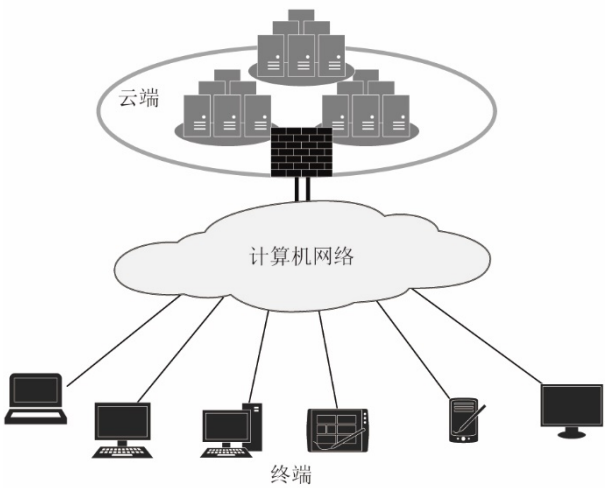


图 5-26 接入云端

负载就是任务，即租户要在云端完成的任务。把众多租户要完成的任务

如何合理地分配给云端的各个服务器并能快速处理完，就是负载均衡技术所要解决的问题。

理想的使用情景描述如下：

早上 8:30 前，微算公司云端只有 1 号服务器在开启状态，其他众多的服务器都处于睡眠状态。处于睡眠状态的服务器耗电极少，唤醒一台服务器的时间大概是 1 分钟。然后员工陆续到岗，最先来的 20 名员工登录到云端的 1 号服务器上的远程桌面，1 号服务器还可以接入 5 台终端，此时 1 号服务器唤醒 2 号服务器，接下来的 5 人仍然接入 1 号服务器，后面来的 30 人接入 2 号服务器。2 号服务器配置高一些，所以能接入 30 台终端，2 号服务器在适当的时候唤醒 3 号服务器，此后其他服务器的行为与 2 号服务器相同。

下午 18:00，员工陆续下班。下班时员工有两种做法：一是从云端注销，二是不注销而直接关闭终端设备。没有注销的远程桌面保留全部的工作任务状态，比如正在编辑的 PPT、打开的 QQ、还在播放的《少年派》等继续保持，第二天员工再次登录时看到的桌面还是昨天下班前的状态，他可以继续编辑未完成的 PPT、浏览别人发过来的大量 QQ 消息，不过昨天还没看完的《少年派》快播放完了。由于云端检测到终端 30 分钟内没有输入，于是把其桌面转为睡眠，视频播放暂停。以后该员工需要倒回 30 分钟并继续观看。云端没有活跃桌面的服务器在延时 10 分钟后主动睡眠。下午 19:00，还有员工在加班，云端还有 11 台服务器在工作，但是每台服务器只接了 1~5 个不等的活跃桌面。此时云端启动了活跃桌面合并程序，采用热迁移技术，把 32 个活跃桌面合并到 1 号和 2 号服务器上，其他服务器转为睡眠状态。云端服务器的唤醒、睡眠和活跃桌面合并都是自动完成的。

上面的情景中有一个关键的技术：租户登录时把其桌面分配到一台合适的服务器上，这就是负载均衡技术要解决的问题。对“合适”二字理解不同，实现负载均衡的策略也不同，但前提是保证不会降低租户的体验。下面是常用的负载均衡策略：

（1）让唤醒的每台服务器承担尽可能多的活跃桌面。

（2）使每台服务器消耗的计算资源的占比尽可能相等。如果全部服务器配置相同，那么本策略就是把活跃桌面尽可能地平均分配给所有的服务器。

（3）应用软件相同的租户桌面尽可能地分配到相同的服务器上。同一部门的租户使用的软件差不多，所以相同部门的租户桌面尽可能地分配在相同的服务器上。例如，售后服务部的员工一般都使用客服软件、办公软件、邮件、问题库工具等，所以把售后服务部的员工的桌面放在同一台服务器上且作为半平台 PaaS 云桌面，从而节约很多内存。

（4）同一个租户每次登录时，其桌面尽可能地分配到同一台服务器上。这样可以尽量减少桌面漫游次数，从而降低网络带宽的开销。

负载均衡任务由负载均衡器完成，目前既有纯软件的负载均衡器产品，如 LVS，也有厂家推出的硬件负载均衡器，如 F5、深信服、梭子鱼、Radware 等，F5 负载均衡器的价格都在 10 万元以上。LVS 是开源软件，已经集成到了 Linux 内核，其性能优越，结合一台通用的物理计算机，完全可以搭建一台能与 F5 媲美的负载均衡器。直接采用 DNS 轮询或者 iptables 也可以实现简单的负载均衡任务。

5.1.6 集群

负载均衡技术用于解决如何把许多互不相关的小型任务或中型任务合理地分配到不同的服务器上的问题。互不相关的小型任务或中型任务是指任务之间没有关联性，而且只用一台服务器就可以完成的任务。绝大多数个人租户的任务都属于这类任务。对于大型任务，由于一台服务器无法按时完成，所以就要把大型任务拆分成许多中小型任务，然后再分配给多台服务器，由它们协同完成，这就是计算机集群技术所要解决的问题。

对租户来说，由很多台服务器组成的集群系统就像一台超级计算机，不管运行多么复杂、大型的任务，马上就能得到结果，如图 5-27 所示。而具备

同样计算能力的超级计算机价格却异常昂贵，所以当 PC 服务器价格大幅度下降之后，人们热衷于采用廉价的集群系统来完成各种高性能的计算任务。比较明显的例子就是，谷歌公司用几万台服务器组建搜索集群系统，而且服务器都是他们自己组装的。注意：本节讲述的集群是高性能计算集群，目的是完成大型的任务，下一节讲述的容错计算追求更好地完成任务。

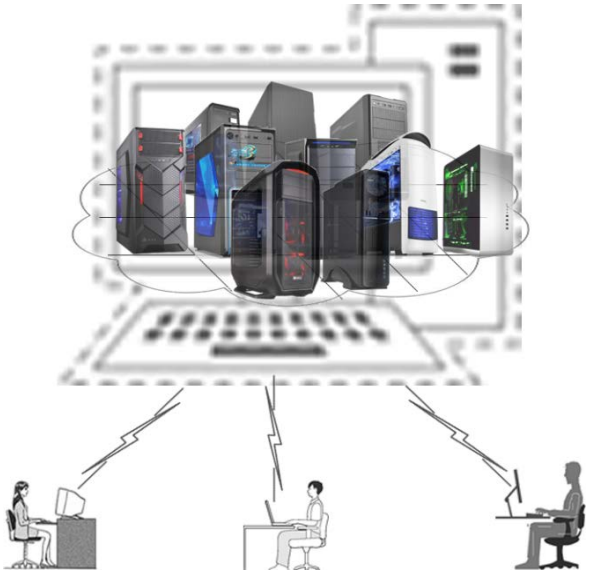


图 5-27　集群构成的“超级计算机”

显然，集群涉及两大技术：一是任务的拆分，二是任务的调度。任务的拆分遵循的原则之一是尽量降低子任务之间的关联性，从而提高处理任务的并行度。最常见的关联性是任务处理的时间先后关系，比如子任务 A 一定要在子任务 B 完成之后才能开始处理，即 B 任务的输出作为 A 任务的输入。存在一类子任务无关联的大型任务，拆分这样的任务就很容易。比如在全球上亿的网页中搜索关键词“云计算”，这个任务的拆分方法之一是按国家划分：A 子任务搜索中国的网站，B 子任务搜索美国的网站，C 子任务搜索英国的网站等，这些子任务之间没有关联性，可以并行处理，最终把全部子任务的搜索结果合并起来即可。再比如中国的天气预报，可以按省份拆分。这类大型任务的拆分如图 5-29 中的左图所示。

另外一类大型任务很难拆分成互不相关的子任务，如核爆模拟、战争推演模拟等，拆分出来的子任务之间一定存在关联性，即一些子任务之间开始或者结束的时间与其他子任务的起止时间存在前后关系，如图 5-28 中的右图所示。

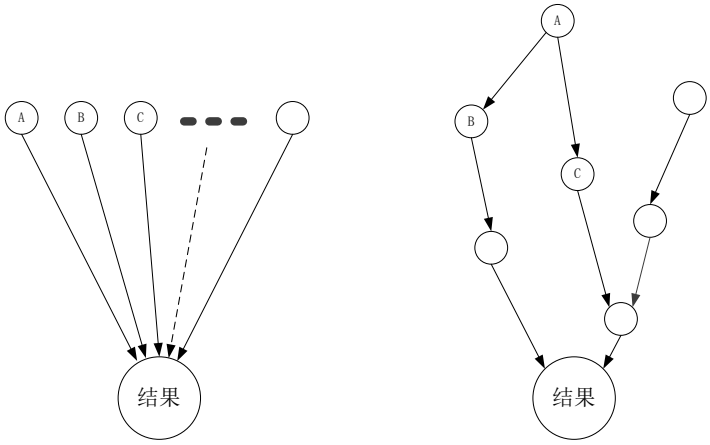


图 5-28　拆分任务

集群涉及的另一大技术是任务的调度，即如何把拆分出来的子任务合理地调度给云端服务器并让它们协同完成，这里的“合理”性与租户的要求密切相关。可以这样定义：最能满足租户要求的调度方法就是合理的。租户的要求各种各样，有的希望任务完成得越快越好，有的要求在将来某个时间前完成任务即可，有的希望结果越精确越好，有的希望计算费用越少越好……根据租户的不同要求，人们提出了相应的任务调度方法，比如在满足租户预期目标的前提下，把任务分配给尽量少的计算机，从而满足“应用最少的计算机按时、按质完成任务”的要求，因为参与的计算机越少，通信的成本就越低。例如，有一个 100 人日的任务，甲方要求在 10 天内完成。为了在规定的时间内完成任务，乙方更愿意安排 10 个人做 10 天，而不是安排 105 个人做 1 天。由于要管理 100 人，所以后者需要增加 5 个管理人员。

集群是一个复杂的工程，它涉及很多分布式方面的基础算法，如选举算法、一致算法、波算法、快照、故障检测等，感兴趣的读者可以参阅 Gerard

Tel 著的《分布式算法导论》。Hadoop 就是一个集群系统，它负责分布式系统的基础算法，从而在 Hadoop 上编写分布式程序就简单多了，详细的介绍请参见后续章节。另外，OpenHPC 是一个基于 Linux 的开源集群项目。

5.1.7 容错计算

容错计算，也有人称为高可用性计算和高可靠性计算，其重点是保证任务在被处理的过程中不会异常终止，以及任务完成后输出结果的正确性。可靠性是指在规定的使用条件下和规定的时间内完成规定功能的能力，通常用多少个“9”来衡量。比如人们所讲的达到 4 个“9”的可靠性，就是说任务顺利完成的概率为 99.99%；同理，5 个“9”的可靠性就是 99.999%。严格来讲，容错计算也属于集群的范畴，只不过参与集群的计算机处理同样的任务——并行或者接力。容错计算需要投入更多的计算资源，所以造价较高。对于一些非关键的任务，人们出于成本的考虑，一般较少采用容错集群系统。

1．接力容错

接力容错又叫串行容错，由若干台计算机参与同一个任务的计算，但是同一时刻只由一台计算机处理任务，只有当这台计算机出现故障时，才由下一台计算机接力处理；类似，如果此台计算机又出现故障，那么继续由其他计算机接力；只有当全部计算机都出现故障时，任务处理才会被中断，如图 5-29 所示。其实我们可以及时维修故障计算机并让其再次加入容错集群参与下一次的接力活动，这样就能最大限度地保证任务顺利完成。只有在全部计算机同时出故障（或者几乎同时出故障以至于人们还来不及维修）的情况下，租户的任务才会被中断，如供电异常（断电、电压过高或过低等）、雷击都可能导致参与容错集群的计算机同时出故障。所以云端的建设，供电和防雷是重中之重，在条件允许的情况下，应采用多路市电接入，每路市电分属于供电部门不同的变压器，甚至是不同的电网，机房配备不间断电源和发电设备，这样就形成了三级供电保障：市电、不间断电源、发电设备。

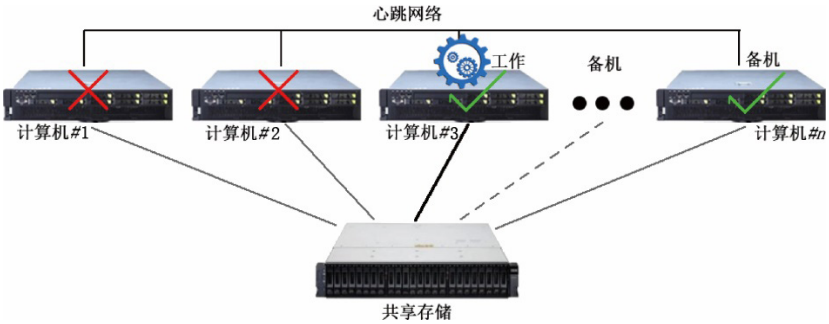


图 5-29　接力容错

传统的民用接力容错集群多采用经典的“双机—双工”结构：两台服务器带一台磁盘阵列，如图 5-30 所示。

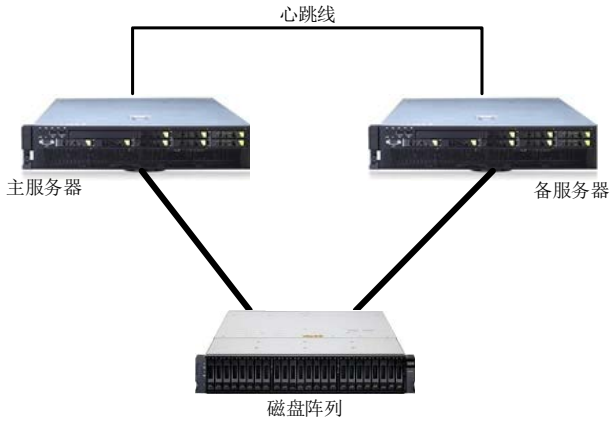


图 5-30　“双机—双工”接力容错

两台服务器都与磁盘阵列相连，同时通过心跳线互联，用来完成任务的软件和软件处理的数据放在磁盘阵列上，这样每台服务器都能访问这个软件，同时双方通过心跳信号感知对方的状态。平时主服务器运行软件以处理任务，备服务器空转，一旦主服务器出现故障，其就不再向备服务器发送心跳信号，备服务器收不到主服务器发来的心跳信号就知道其出现了故障，此时备服务器会运行软件，从而继续处理任务。然后机房管理人员对故障服务器进行维修，而租户并不知道负责任务处理的服务器发生了切换。这种双击容错集群系统被企业大量应用于数据库管理系统、邮件系统及一些诸如 ERP 等重要的

业务系统，以保证关键应用的高可靠性。另外，也有采用“双活”主机的方案——两台服务器平时都承担负载，当一台服务器发生故障时，负载才由另外一台服务器全部承担，应用不会重启，但是相应时间可能会延长。

也许有人会问：磁盘阵列坏了怎么办？在图 5-30 中，磁盘阵列成了单点故障，为此有人采用存储局域网替换磁盘阵列，并在存储局域网中进行存储容错处理，这种方法成本很高。在以太网网速达到万兆甚至十万兆的今天，去掉磁盘阵列而对服务器上的硬盘直接做实时同步变得可行，为此笔者发表了论文《存储寄存算法及其应用》，并在实验室验证通过，试验中采用 DRBD 块同步软件。主/备服务器的硬盘数据相互寄存，如图 5-31 所示。

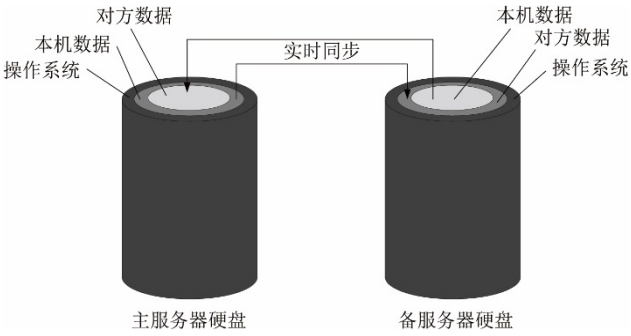


图 5-31　存储寄存

2．并行容错

并行容错是指，参与容错的计算机同时处理相同的任务，输出相同的结果。可以在服务器内部的部件层次做并行，也可以在服务器层次做并行，前者主要由服务器生产厂商设计和完成。现在的绝大多数服务器都或多或少地存在一些并行部件，如双电源供电、多网卡捆绑、RAID 1 支持等。至于在服务器层次做并行，普通民用领域很少采用，多见于航空、军事领域。并行容错如图 5-32 所示。在 3 台计算机参与的并行容错集群系统中，不但可以容错，而且还可以纠错。比如在由 5 台计算机组成的并行容错集群系统中，有 4 台输出 A、1 台输出 B，那么根据少数服从多数的原则，最终的结果就是 A。

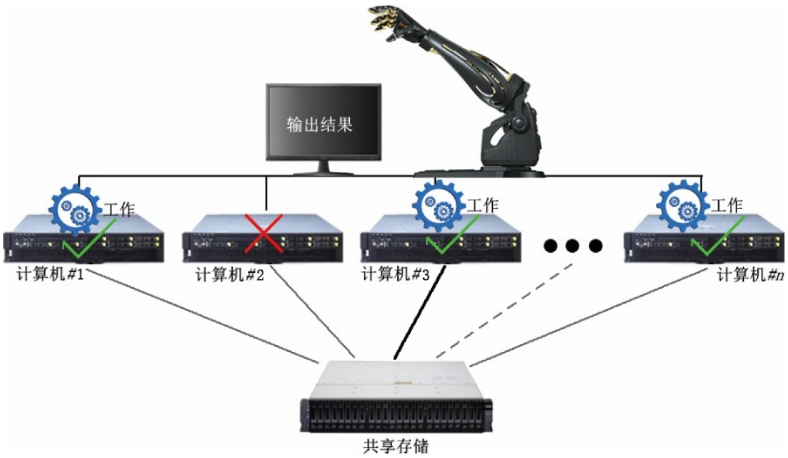


图 5-32　并行容错

例如，由 5 台计算机组成并行容错集群系统，它们接受相同的输入信息并运行相同的决策软件。在运行的过程中，有 1 台计算机出现故障（如图 5-33 所示，中间那台计算机出现故障），最终有 4 台计算机输出结果，其中 1 台计算机给出“不按”下原子弹起爆开关的命令，其他 3 台输出“按”下起爆开关的命令，协商的结果是给机械手发出“按”下命令。

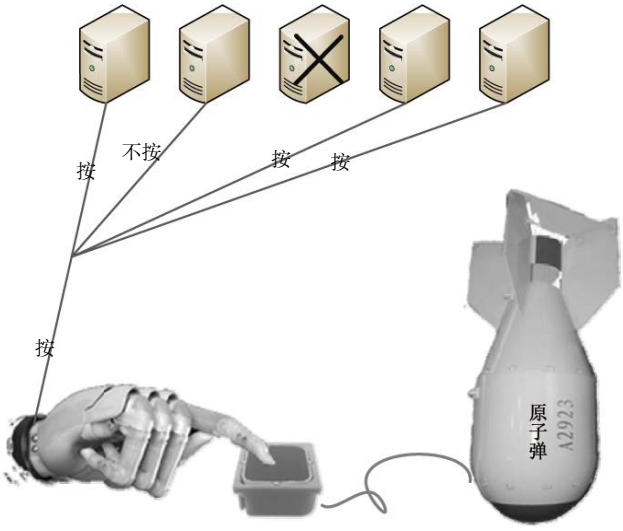


图 5-33　中间计算机出现故障

5.1.8 家目录漫游

用户登录计算机后，首先进入的那个目录就是其家目录。例如，在Windows 操作系统上，zsan 这个用户的家目录是 C:\Users\zsan；在 Linux 操作系统上，其家目录是/home/zsan。家目录下还有更多的子目录，如“计算机桌面”子目录、“我的文档”子目录、“下载”子目录、“我的音乐”子目录、“我的视频”子目录、“我的图片”子目录等。除非特别指定，否则用户的资料都保存在其家目录（或者家目录中的子目录）下。租户每次登录云端，都可能被分配到不同的服务器上，比如某租户第一次被分配到三号服务器，并编辑了一份个人简历，过了几天，他再次登录云端时被分配到七号服务器，此时他的个人简历不见了，因为简历保存在三号服务器上。为了解决这个问题，就引入了租户家目录漫游技术，即磁盘文件作为软件的输入/输出设备之一，与计算设备进行计算机网络分隔，这样计算设备与输入/输出设备就完全分离了，家目录漫游如图 5-34 所示。

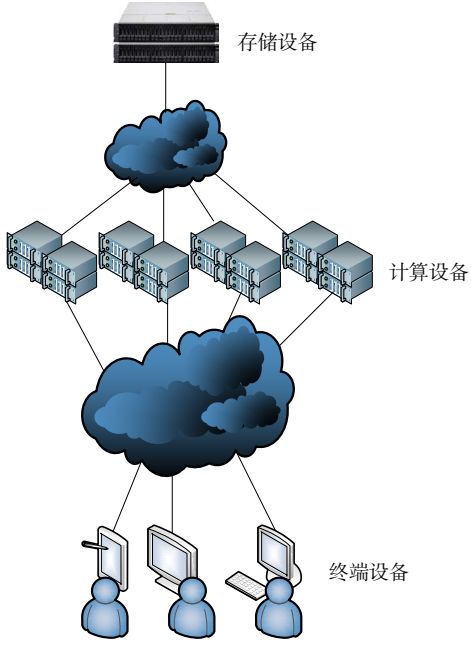


图 5-34　家目录漫游

所有租户的家目录全部存放在网络中的存储设备里，在租户登录时自动触发一个动作：把此租户的家目录挂载到本服务器上。租户退出登录时触发另一个动作：卸载家目录。这样不管租户登录到哪台服务器，他都能正确访问自己的家目录。存储设备在网络上，因此也可以把它当作云端，只不过这个云端的功能是提供存储服务，显然是属于 IaaS 云。

在网速足够快的情况下，建议由政府组建一个公共存储云，每个公民免费分配 1TB 的存储空间，存放个人的全部身份信息、档案材料和私有资料。计算服务云（IaaS、PaaS、SaaS）由企业或者行业主导，它们互相竞争，为租户提供各种各样的计算服务，而租户通过移动或者固定终端随时接入这些云端并获取相应的计算服务。这是一个庞大的系统，需要全社会参与，公共存储云如图 5-35 所示。

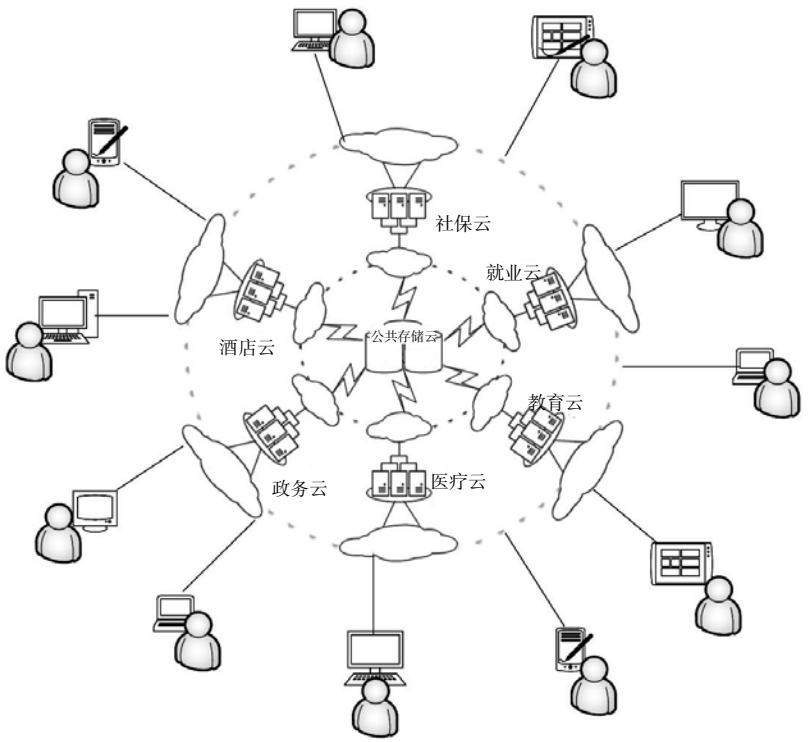


图 5-35　公共存储云

5.1.9 租户隔离

租户隔离不是指隔离租户这些现实中的人，而是特指租户登录云端后，其操作行为和数据对于其他已登录云端的租户来说是不可见的。换句话说就是，每个租户都感觉不到他人的存在，似乎就是自己一个人在操作计算机。存在两个方面的隔离：一是租户行为隔离；二是租户数据隔离，数据隔离比行为隔离更重要。

1．租户行为隔离

租户行为隔离是指一个租户操作计算机的行为，其他租户感知不到。租户操作计算机的行为是通过消耗的计算资源（内存、硬盘、CPU 和网络）体现出来的，换句话说，一个租户消耗计算资源的变动不会引起其他租户计算资源的可感知变动。例如，租户甲内存配额是 4GB，但是他最多也就使用了 2GB，另外 2GB 空闲。此时如果租户乙运行一个大型软件，消耗了很多内存，使得租户甲只剩下 1GB 的空闲内存，但是租户甲感知不到，因为他的软件运行照样流畅、响应速度照样快、网速不卡，一切如故。

现代很多虚拟机厂商倾向于采用如下的行为隔离原则：按实际使用量分配资源，但不超过租用上限。这里的“上限”是指租户租用的资源额度，比如租户甲租用的内存额度是 2GB、硬盘额度是 20GB。如果他实际要消耗 1.5GB 的内存，就分配 1.5GB 给他，但是他最多只能用 2GB 的内存。这样，同样的一台服务器就能服务更多的租户。比如一台服务器拥有 64GB 的物理内存，假设每个租户租赁 2GB，那么这台服务器很可能允许 45 个用户同时登录。

SaaS、PaaS 和 IaaS 模式都需要实施租户行为隔离。

2．租户数据隔离

在“2.6.3 SaaS”一节中，我们提到租户数据包括配置数据和业务数据。配置数据指租户选择的语言、设置的时区、桌面背景图片、屏幕分辨率、创建的快捷方式及各种软件的界面设置等，而业务数据就是日常操作计算机生成的数据，如个人简历、售前 PPT、邮件、音乐、视频、财务数据、库存记

录、客户资料等。

租户数据一般保存在家目录或者数据库中，而家目录和数据库被保存在云端的磁盘中。PaaS 型租户的数据隔离一般采用容器或者操作系统的访问控制列表（ACL）的形式，主要在操作系统层设置；而 SaaS 型租户的数据隔离主要在应用软件层及以上展开，租户身份鉴别和权限控制策略由应用软件开发者负责。比如一个 SaaS 型的 ERP 系统，账号、密码及权限都被登记在数据库中的一个表中，当租户登录 ERP 系统时，输入账号和密码并单击“登录”按钮后，软件要去查询数据库确认账号是否存在；如果存在，再核对密码是否正确；如果密码正确，再根据权限显示相应的模块菜单项。总之，租户的账户信息一定是租户数据记录的主索引的组成部分，这是实现数据隔离的必要条件。

SaaS 型租户数据一般全部保存在数据库中，对于同一个数据库管理系统的租户数据隔离有三种方法可选：一是分离数据库；二是共享数据库但分离 Schema；三是共享数据库和 Schema。

我们用图 5-36 来表示一个数据库管理系统内数据库、Schema、Login、User 的关系。

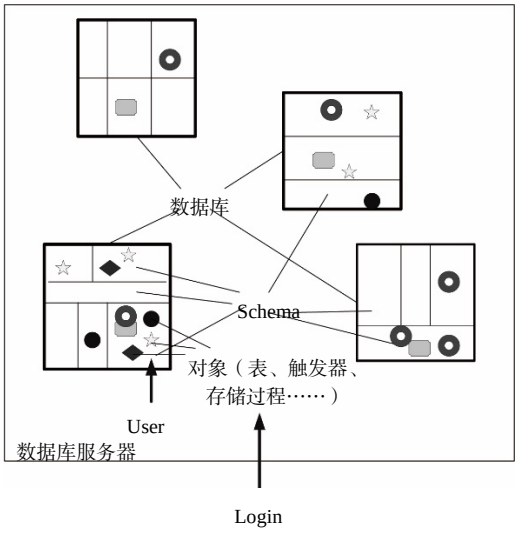


图 5-36　数据库管理系统内数据库、Schema、Login.User 关系

一个具备 Login 权限的人可以登录数据库管理系统（就好像进入了一个大院子），数据库管理系统里面有很多数据库（类似院子里的仓库），数据库被分割成许多个 Schema（类似仓库里的房间），只有持有 User（类似房门钥匙）身份的人才能进入属于自己的 Schema（房间）。Schema 里存放了很多对象（就像房间里存放的桌子、床铺、凳子、电视机等物品），这些对象指表、触发器、视图、存储过程等。理解了数据库管理系统、数据库、Schema 的关系之后，我们再来看看前面讲的三种分离方法。

（1）**分离数据库**：就是给每个租户单独创建一个数据库，数据库中只有一个 Schema，相当于在大院子里单独建一栋只有一个房间的仓库，仓库钥匙只给此租户一个人。

（2）**共享数据库但分离 Schema**：就是在一个数据库中单独为每个租户创建一个 Schema，相当于在一栋现存的仓库里隔一个房间出来并分配给租户，房间钥匙只给其一人。

（3）**共享数据库和 Schema**：直接给租户分配一个现存的 Schema，大家共享一个 Schema，即租户的数据保存在相同的表中，相当于配一把房间的钥匙给新来的租户，大家共用一个房间，每个人的物品上写上自己的名字，以免拿错东西。

上面三种方法中，分离数据库的隔离效果最好，但是成本也最高；共享数据库和 Schema 的隔离效果最差，但是成本也最低；共享数据库但分离 Schema 是一个折中的方法。可根据租户的需求确定具体采用哪种方法。

下面以笔者曾参与云化的一个大型软件为例，来简单介绍一下 SaaS 型租户的数据隔离策略。

该软件原来是落地运行的，主要用于企业产品质量保证，由于价格昂贵，很难推广到广大的中小型企业，所以考虑云化，然后以 SaaS 形式出租。在此项目中，笔者负责撰写云化方案和云端架构设计。在云化方案中，关于租户数据隔离部分设计如下。

采取“独立数据库、共享 Schema”的混合方法，即每个行业对应一个独立数据库，同一个行业内的租户共享 Schema，混合数据隔离如图 5-37 所示。

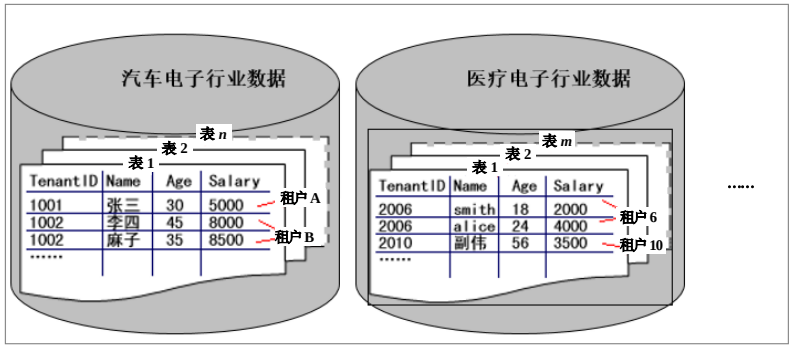


图 5-37　混合数据隔离

混合数据隔离方法具有如下好处：

（1）方便针对各个行业的特点做定制化。

（2）一方面避免表记录数过于庞大，另一方面避免数据库连接成为瓶颈。

（3）便于以后扩展（增加新的行业）。

（4）便于做数据挖掘和商业智能分析。

每个数据库中包含两类数据：一类是共享数据，即全部租户共享使用，如基础数据、全局参数、模型数据等，共享数据表不用改动；另一类是业务数据，这类数据与租户关联，不同租户的业务数据是不同的，由于租户的业务数据是共享表结构的，所以全部的业务数据表都需要添加一个租户字段 TenantID，且此字段作为主键之一。

同时，还要修改用户登录逻辑，在会话中记录用户所属的租户号和行业数据库，最后需要修改全部的 SQL 语句和存储过程，在 where 子句中都要加上“TenantID=xxx”过滤条件，其中“xxx”是具体的租户 ID 号。

如果出现下面的情况，则表明没有隔离：

（1）能看到其他租户运行的软件。

（2）能看到其他租户保存在磁盘上的文件。

（3）当其他租户运行大型软件时，能感觉到计算机卡顿。

（4）当其他租户使用迅雷下载电影时，自己打开网页的速度很慢。

IaaS 云服务（包括裸机、虚拟机和容器），平台软件层及以上都是由租户自己安装和管理的，所以 IaaS 云服务天生就具备了很好的隔离效果。我们在搭建 PaaS、SaaS 云服务时，就要考虑租户隔离问题了，目前没有统一的隔离方法，需要根据租户的需求和性质综合考虑。但是租户隔离不是必需的，比如私有云可以不隔离，因为租户之间互相认识，或亲属，或同事，对于不应公开的资料，自己加密保存或者存储在 U 盘上，此方法既方便又可靠，且节约成本。

对于租户需要输入账号和密码才能登录的公有云服务，必须施行租户隔离。

5.1.10 统一身份认证

以前去井冈山游玩，里面的每个景点都要单独排队购票，很烦琐；现在已改为通票，上山时只需购买一张通票，就可以直接进入任何景点游玩，在每个景点门口出示一下通票即可，一路畅通无阻，既方便又省时。

什么人可以进去、进去后可访问什么资源，以及事后承担什么责任，这是人类社会每天都在发生的事情，如景区参观、厂区上班等。

云端包含很多应用系统，而且租户的家目录还要漫游，统一身份认证就相当于景区的通票，登录云端时只需一次验证，之后就可以进入任何有权进入的应用系统。统一身份认证与单点登录是同一个概念。如果没有统一身份认证，那么你进入任何一个应用系统都会被要求输入账号和密码，这样一方面难以记住众多的账号和密码，另一方面使用云端资源很烦琐。

对于云端应用系统的访问控制，通常分为三个层面：一是认证（解决能

否进入的问题），二是授权（解决进入后能做什么的问题），三是记账（解决做了事情后就要承担相应责任的问题，可能还要付费），这就是通常所说的 3A 安全机制（Authentication，Authorization，Accountability）。基于 3A 安全机制的访问控制在现代操作系统中被普遍采用，例如 Linux 操作系统访问控制步骤如下：

（1）用户输入账号和密码想要登录系统，此时操作系统会进行认证（Authentication），即核对输入的账号和密码与保存在系统里的账号和密码是否相符。如果相符，则允许登录。

（2）登录后的用户并不能为所欲为，其每一步操作都必须被授权（Authorization），比如允许进入什么目录、哪些文件允许读、哪些文件允许写、哪些文件允许删除等，都处于操作系统严密的监视之下。

（3）用户的全部操作都被作为日志记录下来（Accountability），方便以后落实责任、事后监督，并作为付费的依据。

Windows 操作系统也采用相同的方法。

作为云端的统一身份认证系统，必须实现以下四个功能：

（1）**统一用户管理（Identification）**。租户的账号、密码等信息集中存储，统一管理。

（2）**身份鉴别（Authentication）**。当租户想要登录某个应用系统时，验证他的票据或者身份是否合法。

（3）**权限控制（Authorization）**。规定允许登录系统的租户具备哪些操作权限。

（4）**操作日志登记（Accountability）**。记录租户的操作行为，以便事后责任追溯。

有了统一身份认证，租户登录云端并访问应用系统的过程，如图 5-38 所示。

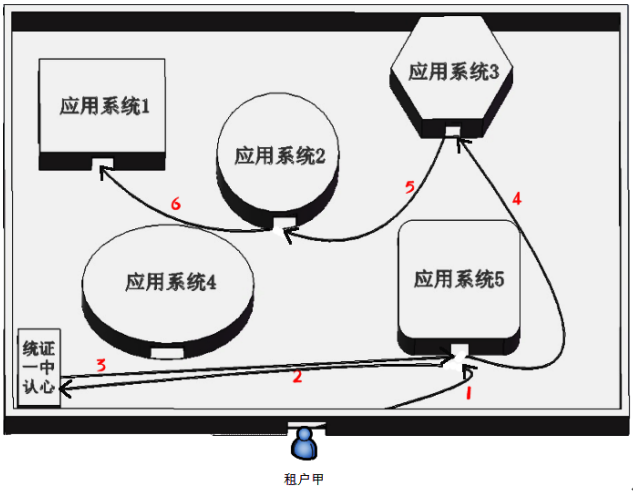


图 5-38　租户登录云端并访问应用系统的过程

租户甲首次登录云端的应用系统 5（第 1 步），但被告知要先去统一认证中心获取票据（第 2 步），拿到票据之后返回并访问应用系统 5（第 3 步），然后凭票据直接访问应用系统 3（第 4 步）、应用系统 2（第 5 步）、应用系统 1（第 6 步）。租户在访问每个应用系统时，应用系统都会查验他的票据，只有票据合法才被允许进入。应用系统在查验票据时都会与统一认证中心确认，不过这一切都是自动的，租户自己感觉不到，但当租户企图访问没有被授权的应用系统时，就会被告知“没有权限”。

不管租户最先访问哪个应用系统，只要租户没有票据或者出示的票据已经过期，都会引导其先去统一认证中心获取票据。但需要注意的是，对租户来说，获取票据的动作只是在屏幕上输入账号和密码，账号和密码的验证、票据的发放等操作都在云端后台自动完成，此后该租户再访问其他应用系统时，就不会在屏幕上显示输入账号和密码的登录界面，因为云端后台自动帮他出示了合法的票据。

5.1.11　SOA

SOA 是面向服务的架构，即企业的 IT 系统是由服务组成的，也即企业的

各个应用系统是由许多标准的服务件“组装”起来的，组成应用系统中的各个服务之间是一种非常松耦合的关系。服务基于简单的“问/答”模型——我问你问题，你给我答案，那么对于“我”来说，“你”就是“服务”。但是答案反馈有同步和异步之分，同步就是我问你问题并在线等待你答复，而异步就是我问完你问题就去忙其他事情了，你有了答案之后再通知我。

在软件行业，基于这种服务的编程思想最早表现为函数，即把经常用到的代码块定义成一个函数并取一个函数名，再用函数名替换程序中原先的代码块（称为函数调用）。比如通过三条边计算三角形的面积，这个任务包含复杂的数学公式，涉及很多条指令，我们可以把它定义为函数 sane(*x*,*y*,*z*)，然后在程序需要计算三角形面积时直接调用这个函数即可。比如 sane(10,20,25)，就会返回边长为 10、20、25 的三角形的面积。

后来人们觉得函数还不够灵活，就提出了模块，模块比函数功能更强，程序就是由模块组装起来的。当然，编写具体的模块时会继续采用函数。模块本质上就是一组类库（一个类库文件包含若干定义好的功能较为强大的函数），允许软件开发人员调用类库中的函数。一些好的面向特定应用领域的模块以开源或者商业模式对外发布，世界各地的其他开发人员可以直接利用这些模块来开发自己的应用程序，从而减少大量重复性的代码编写工作，这样的模块人们习惯称之为框架。例如，利用 Python 语言开发网站的框架 Django，它就是一组类库，编程人员必须掌握需要用到的每个类库函数的定义，才能快速开发网站。

无论是函数还是模块，都要求在程序运行前“组装”好。一旦“组装”完成，函数或模块就静态地捆绑在一起了，所以人们还是觉得不够灵活，于是又提出了运行库的概念。

Gartner 公司在 1996 年进一步提出了 SOA 的概念，意为面向服务的架构，本质上是面向服务的思想在企业 IT 架构方面的应用。面向服务的思想，是面向对象思想之后的一种新的思维模式，其核心特征就是以松耦合、粗粒度的服务单元来构建软件。作为一种思想，SOA 不涉及任何具体的技术实现细节，

但是思想终归要落地才会带来社会效益。人们发现，企业服务总线（简称为ESB）是实现 SOA 的主要技术之一，于是 ESB 也就成为 SOA 的核心技术基础。当然，不用 ESB 也不能说你的系统就不是 SOA，比如现在流行的微服务就是 SOA 的一种具体实现，它采用容器对服务打包。SOA 所实现产品的核心任务是管理企业中的服务单元，具体的任务可分解为：服务单元的登记、服务单元的调用、服务单元的运行、服务单元的部署、用户管理界面，以及安全控制等。

服务与模块的主要区别在于：模块相当于汽车发动机的零配件，而服务就相当于发动机本身，发动机可以独立运转，而零件就不行。函数一般由开发语言编译器的公司提供，如 C 语言编译器有微软的 Visual C++、Borland 公司的 Borland C、开源组织提供的 GCC 等，框架一般由软件开发厂商或开源组织提供，如 Django、Drupal、JSON、Spring、jQuery 等，而服务一般由运营商提供。

企业的软件应用系统和服务的关系像极了人类社会中的项目和人的关系，企业要实施一个项目，先去人才网站招聘各种人员组建团队，然后团队成员各司其职，共同完成项目。求职者事先要在人才招聘网站注册并发布简历，然后等待招聘电话。那么在 SOA 中，也有一个类似人才网站的机构，服务必须先在这个机构里注册，当有需求的时候，其他服务或者应用系统就会在这个机构里搜索能满足需求的服务，并且调用这些服务来完成某个任务。服务像孙悟空一样具备分身术，即同一个服务能分身出很多个体，这些个体分别被其他服务调用，这一点又与现实生活中的求职者不同。

服务是无状态的，即服务在被调用前后本身没有变化，且同一个服务允许同时在多台计算机上运行，这样就能轻松实现高可用性计算及负载均衡集群，如图 5-39 所示。

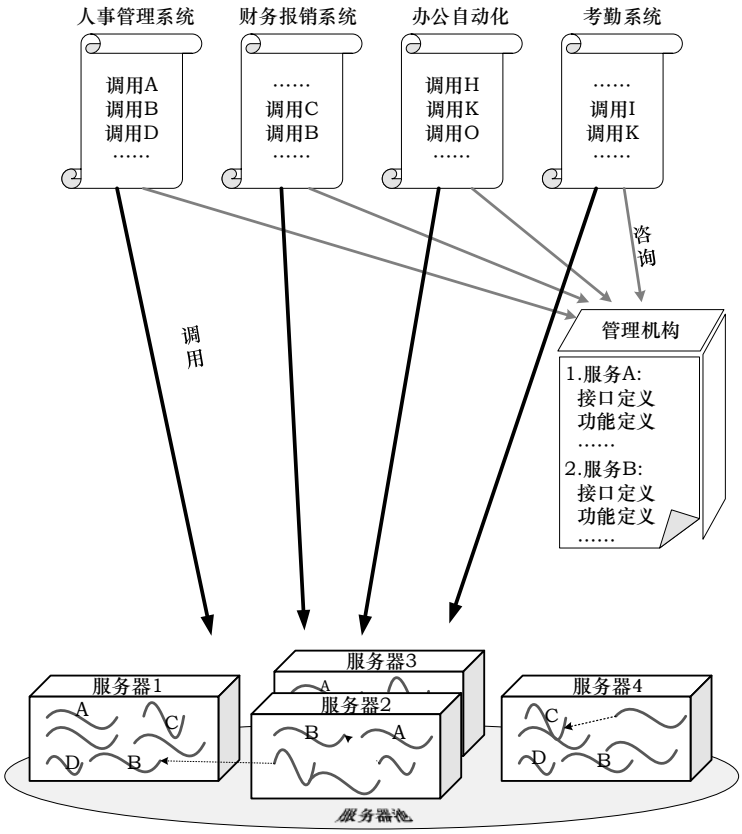


图 5-39　高可用计算及负载均衡集群

最终我们可以想象一下，企业的很多台服务器上运行着各种各样的标准服务，众多的应用系统对应各自的服务调用关系描述表，“组装”一个应用软件由公司文员即可快速轻松地完成。

在云端，由于应用繁多且由一家公司运营，所以云运营公司是采用 SOA 的最佳场所。可以预计，在云计算时代，SOA 将得到广泛应用。在业界，也有人认为云计算将是 SOA 的终结者，这个观点把不同层次的东西混为一谈，云计算不是新的技术和思想，它只是人们使用计算资源的一种模式，而 SOA 是一种全新的软件构架思想。ESB 是实现 SOA 的主要技术之一，SOA 是组建大型云端的重要思想之一。SOA 的概念已经提出十几年了，之前在传统的企业很难落地生根，根本原因就是一家企业的各种软件系统往往是由不同的软

件公司开发的，而这些软件公司具备竞争关系，各自为政，很难协调一致地推出符合 SOA 标准的通用解决方案。

SOA 的实施就是以程序员的视角把公司的业务拆分成一个个服务标准件，然后再选择合适的标准件“组装”成各种应用软件系统。有了一定规模的服务标准件库后，“组装”应用系统就显得异常简单快捷了。在一个高度智能化的 SOA 环境中，“组装”本身也是自动化的。一个公司的软件应用系统越多、越复杂，实施 SOA 就越有价值，因为每个服务标准件得到重复使用的概率就越高；相反，如果一个公司只有一个软件系统，而且以后也不会使用更多的应用系统，那么根本没必要采用 SOA。

“拆分”工作是实施 SOA 成功与否的关键，拆分视角、拆分原则和拆分粒度（服务标准件的大小）必须事先科学规划。最小的拆分粒度就是一个服务对应某种编程语言的一条语句，这时“组装”软件系统就是传统的软件开发，由软件开发工程师来完成；最大的拆分粒度就是一个应用系统对应一个服务，这时“组装”软件系统就是安装一个应用软件系统，由系统管理员来完成。

科学的拆分粒度介于这两种拆分粒度之间，拆分出来的服务标准件既具备一定的可重用性，又能大大简化“组装”程序的复杂性。通行的做法是，先在软件系统的传统“三层”分层结构的基础上，纵向划分出更多的层次，然后对每一层结合企业的业务特点横向划分出多个“块”，每个“块”完成少量的业务逻辑，这样的“块”就是一个服务标准件。拆分的过程是一个不断优化的动态过程，也是一个由粗粒度到细粒度不断演化的过程。

软件的传统“三层”分层结构是指展现层、业务逻辑层和数据层。为了理解它们，请看下面的例子。

王老师打开学校的考试网站，然后查看张宇法同学的“云计算”课程综合考评分数，屏幕上显示 89 分，软件的传统“三层”分层结构如图 5-40 所示。

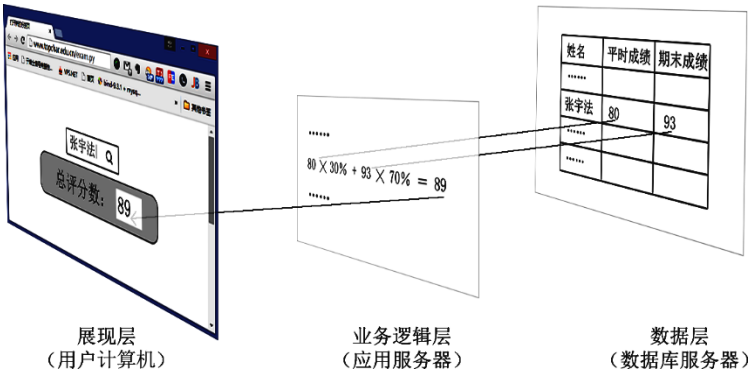


图 5-40　软件的传统“三层”分层结构

数据层里保存着原始的数据（比如平时成绩和期末成绩），业务逻辑层根据具体的业务需求对数据层里的数据进行加工处理（比如平时成绩按 30%、期末成绩按 70%计算总评分数；学工处分配助学金时只看期末成绩，即平时成绩按 0%、期末成绩按 100%计算），展现层决定以什么样式来显示业务逻辑层加工处理的结果（比如以白底黑字宋体 30 字号显示“89”）。

在设计数据层时，重点考虑库表结构；在设计业务逻辑层时，根据具体的业务需求编写加工处理算法；而在设计展现层时，重点考虑屏幕上的内容布局和显示样式。业务逻辑层本质上就是把数据层中的数据映射到展现层上，到底如何映射，与具体的业务有关。

在现实中，有的程序员喜欢使用存储过程把部分或全部的业务逻辑算法放入数据层，这样做有一个明显的缺点，即难以对系统做横向扩容来满足日益增长的用户访问量。因为数据层本身带有状态，对带有状态的系统扩容最常用的办法是纵向扩容，而纵向扩容比横向扩容成本更高、难度更大，因为纵向扩容目前的方法就是给单独的服务器增加更多的 CPU、内存等计算资源。

目前部署 SOA 的应用环境有开源产品和商业产品，开源产品有 WSO2、Dubbo 和 Mule ESB，后者侧重于企业服务总线，不是一个完整的 SOA 套件，这三个开源产品是用 Java 语言开发的，另外一个 ZATO 开源项目是采用 Python 语言开发的；商业产品有 Oracle SOA 套件和 IBM SOA 基础栈等。

5.1.12 微服务

微服务是 SOA 的一个简化版本，并且是具体的实现技术，采用容器对服务打包，可以这样说，如果没有容器技术，微服务就发展不起来。我们都知道，传统的单体应用程序会随着功能的扩展变得越来越庞大，最后修改代码、版本升级或者重新部署都会变得异常困难，甚至根本无法进行。微服务的出现就是用来解决这个问题的——把一个庞大的单体应用横向切割成若干个微服务，每个微服务只做一件事，但它仍然包含展现层、应用层和数据层。微服务单独运行，对外暴露 API 接口供其他程序调用。所以说，微服务侧重于替换企业内部的大型单体应用，以便应用程序的可持续演进（持续代码完善、持续版本升级、持续缩放部署、DevOps）。由于每个微服务都有自己的数据层，所以这个带有状态的微服务就很难跨应用调用。由于每个微服务只做一件事，所以复杂度大大降低；另外，微服务可以单独开发和部署；再者，微服务可以单独缩放扩容，这些都是优点。但是微服务也存在不足之处：微服务之间的调用关系更复杂，数据一致性保证更复杂，总体微服务部署更复杂。一个典型的基于微服务的应用部署包括若干个微服务实例、API 网关、微服务注册机构及若干负载均衡器等。

5.2 终端技术

云终端是用户操作的主要设备，也是用户接入云端的入口。追求完美的用户体验是终端厂商和云服务提供商唯一的选择，也是云计算成败的关键因素。为了使一款终端实现“用之赏心、观之悦目”的目标，需要从功能、外观、操作、质量几方面入手。

5.2.1 终端分类

终端在外观上体现为多样性，分类如下。

1．按功能分类

（1）**人/机交互终端**：不断地通过键盘、鼠标、触摸屏、声控等方式输入信息，通过显示器、打印机、音箱、发光灯等设备输出计算结果，人机协同完成各种任务，如各种台式云终端、智能手机、平板电脑等。

（2）**机/机交互终端**：生活中的各种智能家电、工厂里的智能流水线、工业机器人等，这些机器本身与嵌入的云终端直接互动，完成控制和信息收集任务。

（3）**输入终端**：持续采集数据并源源不断地传输到云端，这些数据将在云端存储、计算，如检测仪、数据采集卡、摄像头、人体监护器等。

（4）**输出终端**：云端把计算结果通过这些终端输出，如打印机、机械手臂、导航仪、显示屏等。

2．按移动性分类

（1）**移动终端**：靠离线能源续航，且相对轻便、便于移动，常被人拿在手里、穿在身上、戴在头上、放在移动的设备上。

（2）**固定终端**：相对而言，体积大而且比较重，常会在办公桌上、智能设备中、墙壁上见到它们的身影，如图 5-41 中的左图所示。

（3）**固定/移动两用终端**：终端由几部分组成，这几部分可以轻易拆卸和组装，组装在一起时充当固定终端，用于办公、软件开发和图形处理等；拆卸之后又便于携带和移动，如图 5-41 中的右图所示。

3．按屏幕尺寸分类

屏幕覆盖 1 ~ 50 英寸的各种尺码，按照尺码大小分为以下几类。

（1）**5 英寸以下的移动终端**：主要用于通信、小内容信息收发（短信、微信、微博等）、音乐播放、导航、小游戏、信息采集、远程监控、远程定位、照相等，可随时随地携带和使用，真正体现人机合一的云生活。一般人走动

的时候拿在手里使用。但是这样的移动终端很容易丢失、被盗和损坏，所以资料存放于云端就显得非常重要。

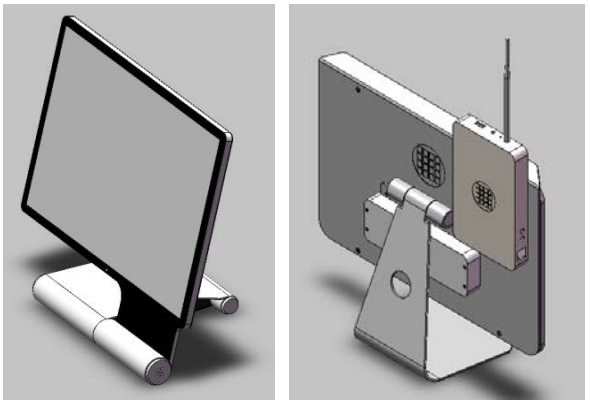


图 5-41　终端

（2）**5～9 英寸的终端**：主要功能包括大内容信息阅读（如网页、邮件、各种文档资料、图片、视频等）、各种弱交互性 APP、非经常性的信息收集（如会议记录、采访记录等），配上其他便于交互的部件（如键盘、鼠标、大显示器等），可以实现日常办公。一般人站立的时候拿在手里使用。

（3）**10～14 英寸的终端**：输入/输出部件分离（如键盘和显示器），能完成绝大多数的日常办公，属于真正的移动办公终端。一般人坐下后放在腿上或桌子上就能投入工作。

（4）**15～18 英寸（不含 18 英寸）的终端**：带触摸屏，常固定，多用于公共场合的信息查询、家庭相册和视屏电话（挂于家庭墙壁上）、老人小孩学习娱乐用机、智能设备的触控、智能会议厅、智慧教室、家庭音乐播放等。

（5）**18～24 英寸的终端**：固定办公终端，替代目前的台式机，完成日常办公、图片处理、音视频剪辑、影视后期制作、软件开发、游戏开发等，配备标准的输入部件。

（6）**25～31 英寸的终端**：带触摸屏，常固定，多用于公共场合的信息查询，不是大众产品。

（7）**32～50英寸的终端**：主要用于娱乐和信息展示，采用遥控输入方法。

5.2.2 质量控制

云终端是用户直接携带和操作的设备，使用环境千变万化，所以质量非常重要。云终端要做成纯硬件设备，终端上用到的软件要全固化或者半固化。全固化是指软件根本无法修改和删除；半固化是指不允许用户自己更改和删除软件，但是允许在线升级、打补丁，并提供出厂配置一键恢复功能。这样就可以大大降低产品的故障率，延长折旧期。

另外，终端中尽量少用软件，甚至只固化一个下载程序（BootLoader），启动时由这个下载程序临时从云端下载经过定制和裁剪的操作系统及会话软件，然后执行这个操作系统，最后运行会话软件，完成登录云端的操作。不过，这种技术要求网络速度足够快。

终极方法是连下载程序都不要，直接采用网卡的网络启动技术，开机时由终端上的网卡直接获取需要的软件并启动终端。

但不管怎么样，一台终端启动后都是要在终端上运行一个客户端程序，这个客户端程序把用户的输入发送到云端，然后把云端计算的结果显示在终端屏幕上。

5.2.3 终端项目

1．RPTC

RPTC的全称是Raspberry Pi Thin Client project，意为树莓派瘦客户机项目。结合树莓派ARM小主板（见图5-42），就能定制一台全功能的低功耗云终端，项目网站为 http://rpitc.blogspot.com/（需要翻墙才能打开此网站）。从此网站可以下载一个安装镜像文件，这个镜像文件采用 Linux 内核，集成了当前绝大多数的协议客户端。截至2018年8月，包括的客户端及其版本如下：

（1）Citrix Receiver 13.4.2。

（2）VMware Horizon 4.3。

（3）NoMachine 5.2.11。

（4）Virt-viewer/Spice 5.0。

（5）Suite3270 3.5ga10 (IBM 云终端)。

（6）FreeRDP（git 上 2017 年 2 月份代码）。

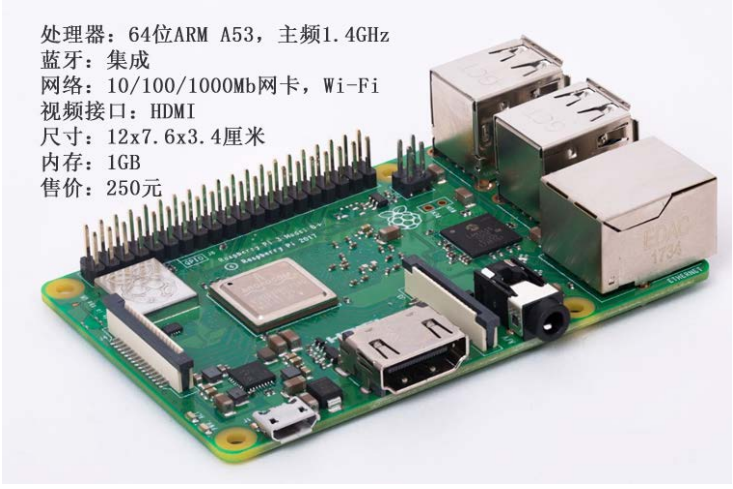


图 5-42　树莓派 3 代 B+

2．Windows 10 IoT Core

这是微软的物联网版本的 Windows 10 操作系统，经过定制可以运行在各种低功耗、资源受限的设备上，支持 ARM 和 X86 架构，如树莓派 2 代和 3 代。Windows 10 IoT Core 是免费的，采用它和树莓派 3 代可以轻松打造一款支持 RDP 10.0 的云终端产品，在线定制网站为 https://developer.microsoft.com/en-us/windows/iot/GetStarted#，也可以从 https://www.microsoft.com/en-us/download 网站搜索 Windows 10 IoT Core 并下载。

3．PiNet

PiNet 项目是一个基于树莓派的电子教室项目，用一台普通计算机做教师

机，里面安装 PiNet，树莓派采用 SD 卡启动，项目的官方网站是 http://pinet.org.uk/。利用 PiNet 创建的电子教室具备如下功能：

（1）集中管理学生账号。学生可以在任意终端上输入自己的账号并登录。

（2）网络启动终端。终端启动时自动从服务器上下载操作系统并启动。

（3）师生共享文件夹。

（4）作业发放与提交。

（5）广播教学等。

PiNet 项目是 Raspi-LTSP 项目的替代品。图 5-43 所示是一个实际的电子教室。

图 5-43　PiNet 电子教室

5.3　通信协议

看过中国香港警匪片的人应该对下面两个警察的对话耳熟能详：

“阿 Sir，我发现他了，要不要跟上？Over。”

“跟上，但不要靠得太近，以免被发现。Over。”

“明白。他进入超市了，那里人多，要不要暗地疏散人群？Over。”

“……Over。”

这样，两个人“Over”来“Over”去的通话，其实就是警察局制定好的通话规则，只要向对方说出“Over”，就表明话讲完了，该轮到对方说话了。通话规则也叫通话协议，规定双方通话时必须遵守的规则。终端与云端通信时相互遵守的规则就是通信协议，双方必须严格遵守事先制定好的通信协议，否则通信无法正常进行。云计算通信协议涉及安全、图像处理、数据压缩、网络传输协议等多个领域，直接决定着租户的终端体验。目前，终端与云端的通信协议主要有如下几种。

1．HTTP/HTTPS

这是网站采用的通信协议，HTTP 的默认端口为 80，明文通信；而 HTTPS 的默认端口为 443，密文通信。密文通信是指，通信双方先加密内容然后再发出去，收到对方的信息后需要先解密。本协议常用于 PaaS、SaaS，以及未安装操作系统前的 IaaS。比如租赁了一台裸机（属于 IaaS），通过浏览器打开裸机的远程管理卡网页，然后通过这个网页给裸机安装操作系统。

2．RDP

这是微软的远程桌面协议，最新版本为 RDP 10，这个版本改进很大，具备这些特征：支持最大 32 位色彩、采用 128 位的 RC4 加密算法、远/近端计算机共享剪贴板、允许远程应用使用本地端口（端口重定向）、本地文件系统重定向到远程计算机、远程声音重定向到本地播放等。RDP 协议规范公布在官方网站上，任何人都可以参照此规范开发遵循 RDP 协议的通信软件。

RDP 10 在 Windows 10 和 Windows Server 2016 上直接支持，Windows 8 和 Windows Server 2012 支持 RDP 8.0，Windows 7 上需要打支持补丁包才支持 RDP 8.0。RDP 也有针对 Android\Mac OS 操作系统的客户端软件，但是用

户体验要差一些。微软发布的远程桌面客户端软件是 mstsc.exe，有安卓版和 Mac 版，但是没有 Linux 版。有一个开源软件 FreeRDP 可以在 Linux 上运行，但是只支持 RDP 7.0/RemoteFX。另外，Ericom 公司推出了多平台版的客户端和具备 RDP 加速功能的服务器端。RDP 协议常用于 PaaS 和安装了 Windows 的 IaaS 云服务。RDP 协议能同时支持远程桌面虚拟主机（RDVH）和远程桌面服务（RDS），利用微软的终端服务和 RDP 技术可以搭建规模小、性价比高的集中办公系统——程序和数据放在服务器上，员工通过终端登录服务器办公。最节约成本的方法是在一台配置高的计算机上安装 Windows 10 和各种办公软件，并打上多用户补丁或者安装 hinstuff XP/VS Terminal Server 多用户软件，然后员工可利用各种终端设备登录 Windows 10，并使用其中的办公软件。

图 5-44 所示是从 Windows 7 登录到云端的 Windows 8 桌面（单击“开始→所有程序→附件→远程桌面连接”或者直接运行命令 mstsc.exe，然后输入云端 IP 地址），采用的是 RDP 8.0 协议。

图 5-44　云端的 Windows 8 桌面

3．PCoIP

这是 EMC 公司（收购 VMware）和 Teradici 公司共同开发的基于 UDP 的

远程桌面协议，意为 IP 上的个人计算机（PC-Over-IP），在低带宽的广域网上也能流畅使用。PCoIP 在主机上做图形渲染，终端只负责解码，因此减少了一些传输量，是专门为瘦终端而设计的。针对 Windows、Linux、Mac OS、Android 和 iOS 操作系统，都发布了相应的客户端软件。

利用此协议搭建的云计算系统结构较复杂，软件许可种类多，这是一个缺陷。图 5-45 所示是一个最小云计算系统，从图中可以看出，必须要有微软的域控 AD、连接器 Connector、虚拟主机（里面安装连接代理）和终端（安装客户端软件）。第二个缺陷是目前还不支持微软的远程桌面服务，但是 Teradici 公司的产品 Teradici Arch 弥补了这个缺陷，该公司同时还推出了 PCoIP 的硬件加速卡。

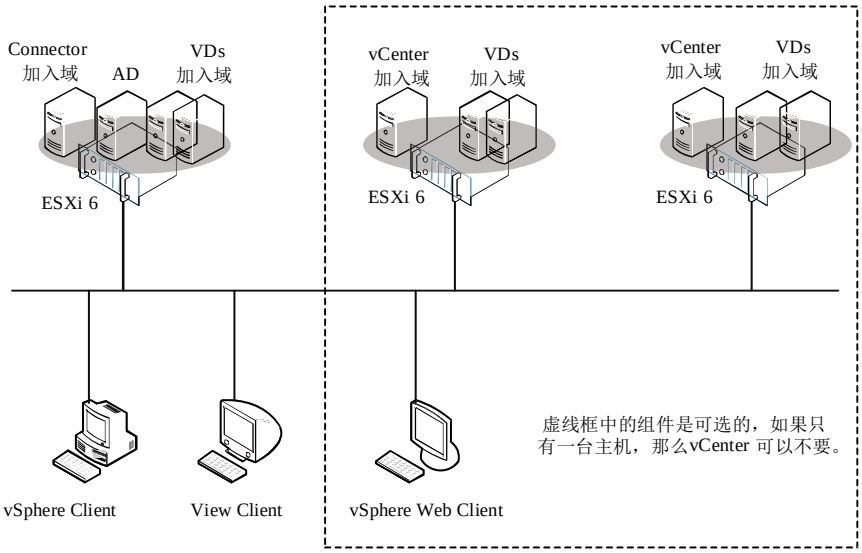


图 5-45　最小云计算系统

必需的组件包括：AD、Connector、VD 和 View Client。VD 可以是虚拟机桌面，也可以是物理机桌面，被 vCenter 管理的桌面为受管桌面，否则称为非受管桌面，VD 里都要安装 View Agent。vSphere Client 是用来管理 ESXi 或者 vCenter 的，如果没有这些，那么 vSphere Client 也可省掉。客户机上必须安装 View Client。vSphere Web Client 只能访问 vCenter。连接的路径是 View

Client ←→Connector←→View Agent。

4．HDX/ICA 协议

HDX/ICA 协议是思杰公司发布的远程桌面协议，采用 TCP，这一点与 PCoIP 采用 UDP 不同。思杰是最早做远程访问的公司，所以其技术积淀比较雄厚，在非常低的网络上（10Kb 左右）也有良好的表现。但不管怎样，采用此协议搭建的云计算系统还是要混合微软的产品的（操作系统、AD 域控等），相比微软的 RDP 协议，其架构较复杂。主要组件包括：外网访问网关（Access Gateway）、桌面交付控制器（Desktop Delivery Controller）、资源池（XenServer Resource Pool）、文件服务器（File Share）、应用控制器（XenApp Controller）、虚拟机镜像供应服务器（Provisioning Server），如图 5-46 所示。

在局域网里面，最简化的采用 HDX/ICA 协议搭建的云计算系统为：终端（安装客户端软件）←→ Desktop Delivery Controller ←→虚拟机（安装 Agent），虚拟机的基础架构可以是 XenServer 6.x、ESXi 6.0 或 Hyper-V。不过，针对远程虚拟桌面市场，思杰公司特别推出了端到端的解决方案包 XenDesktop，目前版本号为 7.6。

5．SPICE 协议

作为新兴的远程桌面协议，SPICE 协议最初由 Qumranet 公司开发，后来被红帽公司收购并开源，且被整合到红帽的云计算产品中。SPICE 首先尝试将渲染的工作交给终端，利用终端的硬件资源来加速，再根据协商的结果考虑是否将渲染工作交给主机来处理，这时可以用软件或者 GPU 来渲染。SPICE 的双向音频流技术支持软件拨号和 IP 电话。SPICE 客户端支持 Android、Linux、Windows 和 Mac OS 操作系统，可以单独从 http://www.spice-space.org/download.html 下载。SPICE 协议包括三部分，分别安装在物理机、虚拟机和终端中，HDX/ICA 协议搭建的云计算系统如图 5-47 所示。

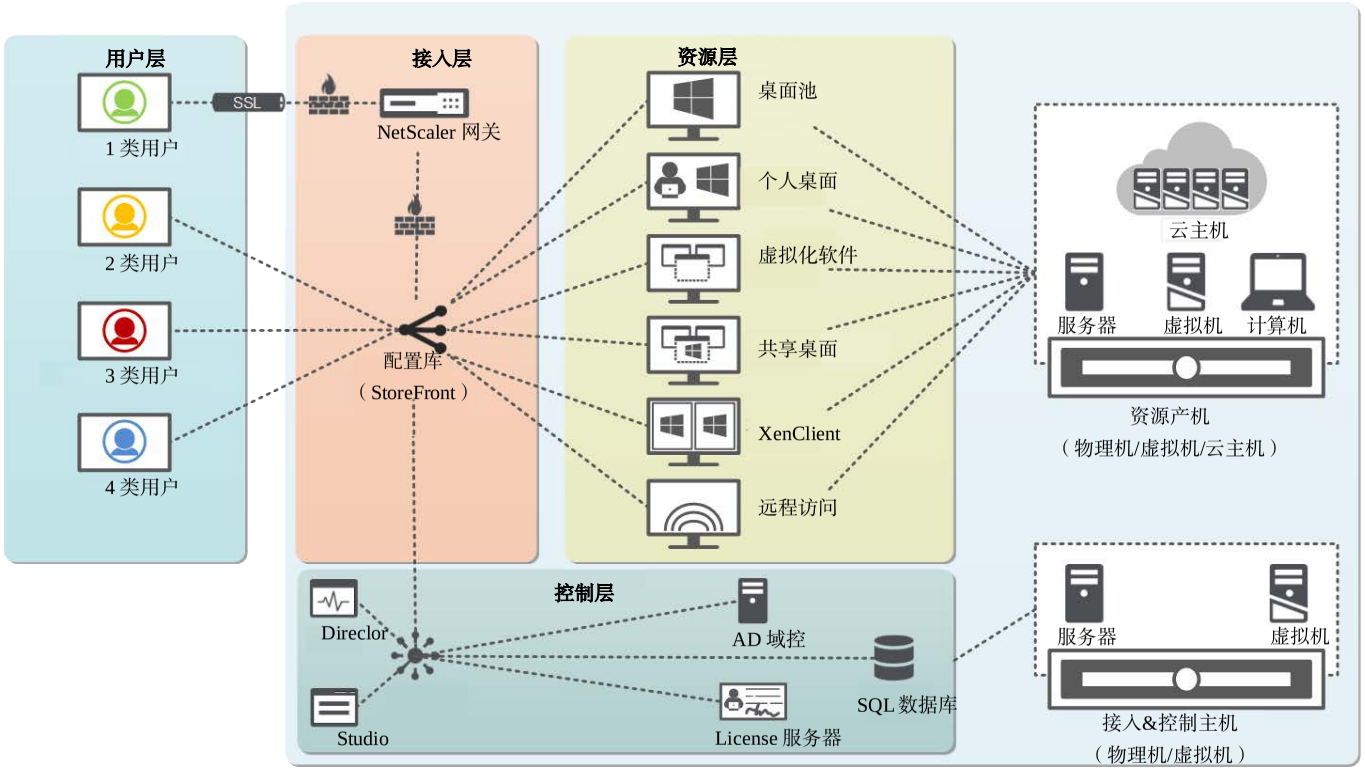


图 5-46　HDX/ICA 协议搭建的云计算系统

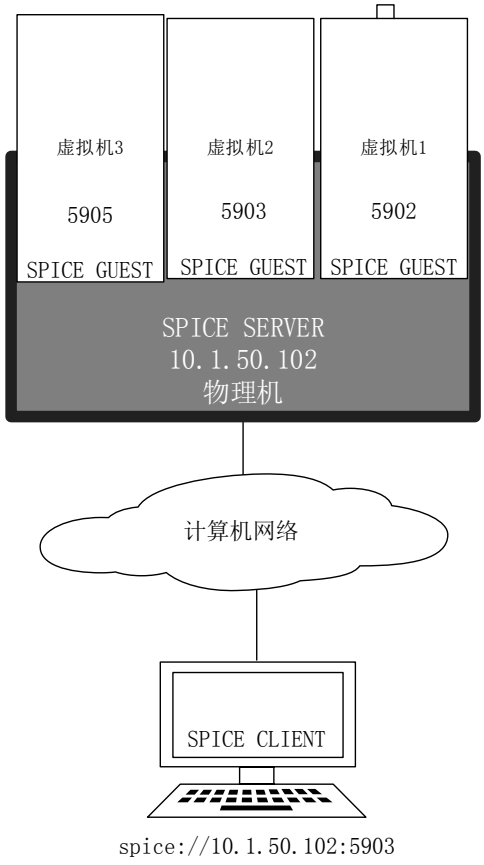


图 5-47　SPICE 通信协议

在图 5-47 中，通过终端的 SPICE 客户端软件打开 spice://10.1.50.102:5903，即可连接到云端的虚拟机 2，如果虚拟机里安装了 Windows 7 操作系统，那么在终端上就可以看到 Windows 7 桌面了。需要注意的是，终端统一访问物理机的 IP 地址，然后由端口号来区分是哪台虚拟机，比如 5903 代表虚拟机 2、5905 代表虚拟机 3。由此可见，SPICE 不支持微软的终端服务，也不支持裸机（不能直接连接到物理机的 Windows 桌面），只支持虚拟机。SPICE 作为一款开源的远程桌面访问协议软件，具备良好的发展势头，尤其是对于国内首次涉足云计算的公司来说，直接集成 SPICE 到自己的云计算产品中，不失为良策。

6．RFB 协议

RFB 协议即远程帧缓冲（Remote Frame Buffer）协议，它直接把控制台的底层屏幕帧的内容实时同步到终端，所以能适应所有的操作系统。但是其传输信息量大，所以一般只在局域网内做点到点连接（一台计算机只与一台终端连接），以便远程协助解决计算机问题。

VNC 是由 AT&T 的欧洲研究实验室采用 RFB 协议开发的最典型的软件，后来在此基础上又衍生出了很多版本，其中 RealVNC、UltraVNC 和 TightVNC 最具代表性。RealVNC 既发布商业版，也发布免费版，不过免费版的功能太弱，而 UltraVNC 和 TightVNC 则完全免费。三款软件的功能旗鼓相当，但是也有缺陷：一是只能看到或者操纵同一个桌面，尽管允许不同的系统账号同时登录，而且这些系统账号必须具有系统管理员权限；二是有的信息（比如 Windows 7 经常提示的“需要管理员权限才能运行此软件”）只能显示在本地的屏幕上。RFB 协议如图 5-48 所示。

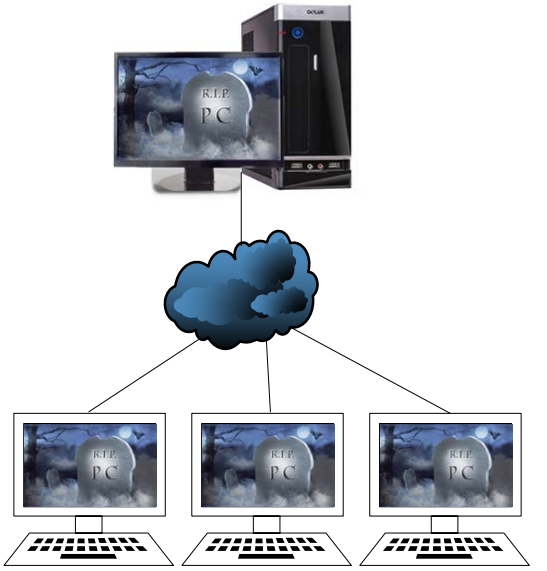


图 5-48　RFB 协议

RFB 协议不太适合云计算，而适合远程协助解决问题和广播教学——老师先进入教师机并启动 VNC 软件，学生以“查看”模式登录教师机，这样学生都能看到教师机的屏幕。

7．X 协议

X 协议是由麻省理工学院开发的开源网络图形显示协议，采用的是客户机/服务器架构。基于这个协议开发的软件有 XFree86 及其继任者 Xorg，被类 UNIX 操作系统广泛采用，作为其图形桌面组件的一部分，比如 Linux 操作系统大多数采用 Xorg。X 协议的客户机/服务器模式如图 5-49 所示。

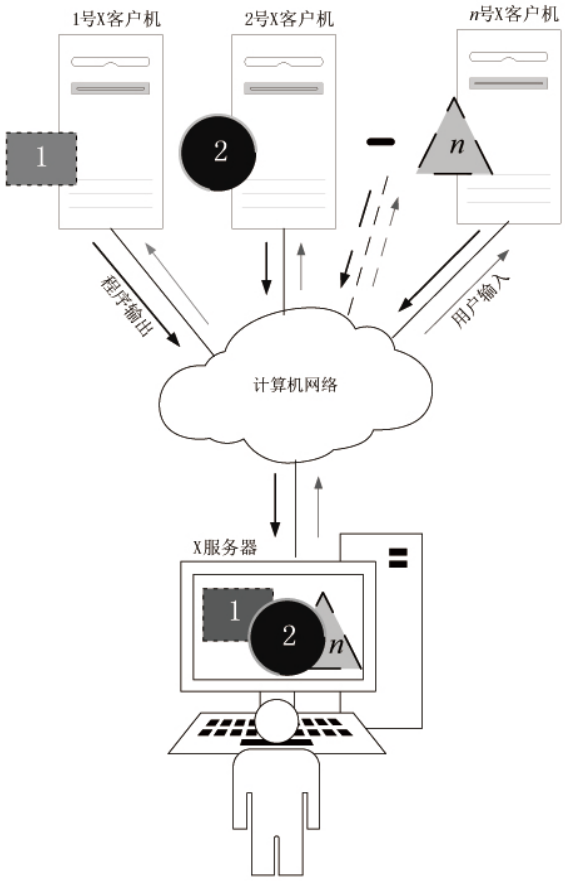


图 5-49　X 协议的客户机/服务器模式

X 协议的客户机/服务器模式与传统的客户机/服务器模式不同，传统的客户机/服务器模式是针对用户而言的，即用户操作的计算机为客户机，远方的计算机为服务器；而 X 协议的客户机/服务器模式是针对程序而言的，即程序运行的那台计算机叫客户机，把结果输出到服务器上，服务器就是用户操作的那台计算机。如图 5-50 所示，1 号、2 号和 *n* 号 X 客户机各运行一个程序，它们分别输出长方形、圆形和三角形，最终都在 X 服务器上显示出来。

令人遗憾的是，X 协议不传递声音，比如在 X 客户机上运行音乐播放器，而坐在 X 服务器旁边的用户是听不到声音的，不过有一些能支持声音的第三方开源项目，如 FreeNX、X2Go、LTSP 等。

利用 X 协议可以打造一台万能终端，即同时接入不同的云端。这一点很重要，因为在云计算时代，没有一家云服务提供商能通吃全部的云服务，一定会存在很多云端，且提供各种不同或者相同的云服务。作为用户，希望只使用一台终端。

PulseAudio 项目弥补了 X 协议不能传递声音的缺陷；开源 LTSP 项目整合了 Linux、Xorg、PulseAudio、OpenSSH 等开源软件，专门用于搭建教育培训机构和中小企业的私有云，终端不用事先安装或者固化任何软件，只要一块支持 PXE 启动的网卡即可，LTSP 的官方网站为 http://www.ltsp.org/。针对树莓派的衍生项目 http://pinet.org.uk/，结合低廉的树莓派小主板，就能轻松搭建教学云。NoMachine 公司发布的 NX 协议早期就是压缩 X11 协议并通过 SSH 安全通道传输，后来公司认为这样做不安全，从 4.0 版本开始默认采用改进的 NX 协议直接传输，当然仍然支持 SSH 安全通道，并且针对企业用户不再开源。

8．HP RGS

惠普远程图形软件（Remote Graphic Software，RGS）是一款高性能的远程图形系统，它允许用户通过标准 TCP/IP 网络访问和共享远程计算机桌面，在远程计算机上完成图形渲染，然后把渲染结果发给客户端。RGS 为客户机/

服务器模式，服务器称为 Sender，客户机称为 Receiver。典型的应用场景是：配置较高（强悍的 CPU、大容量内存、SSD、独立显卡）的远程工作站安装图形处理软件（如 CAD、SolidWorks 等）和 Sender；用户使用的瘦客户机安装 Receiver，然后运行 Receiver 软件连接到远程的 Sender，即可操作那里的应用程序（如 CAD 等）。相比其他协议，RGS 的用户体验要好很多，NASA 与 HP 实验室合作，利用 RGS 的核心压缩技术实现长距离的火星探测，证实了 RGS 绝非等闲之辈。RGS 的一些重要特性如下：

（1）流畅支持 3D 图形。

（2）先进的视频压缩技术（Windows 和 Linux 有效）。

（3）支持 Windows、Linux 和 Mac OS 操作系统。

（4）允许多个 Receiver 同时连接到相同的 Sender，轻松实现协同办公或教学。

（5）运行同一个 Receiver 同时连接到多个 Sender。

（6）采用惠普 Velocity 技术增强跨广域网的用户体验。

（7）服务器上的高保真音频能通过本地喇叭播放。

（8）本地 USB 设备能挂载到远程机器上。

（9）本地设备允许触摸和手势输入。

（10）RGS 发送像素而非数据，因此，无论你在使用什么应用，RGS 都可正常运行。

（11）可自动进行分辨率和多显示屏设置，以匹配客户端设备。

RGS 的应用领域有桌面虚拟化、远程演示、移动办公、协同办公、教学/培训等。

综上所述，在局域网内做比较，从用户体验方面来看：RGS 最好，PCoIP 协议要优于 HDX 协议，HDX 协议又优于 RDP 8.0（Ericom 公司改进了 RDP 8.0，并且终端可以直接采用浏览器），但是 RDP 10 有了很大改进，RDP 10 要好于 SPICE 协议，X 协议和 RFB 协议似乎垫后。

从架构复杂性方面来看：采用 RDP 组建的云计算系统最简化，产品单一、软件许可证单一，小到一台计算机即可做云端，甚至可做服务全球的云端；采用 LTSP 开源项目搭建简单云端比较容易，而且全部是开源软件，但是缺少 Windows 上一些流行的应用软件（如 Photoshop、Microsoft Office 等），如果做类似 iPad 的深度定制化，则是一个不错的方向，Raspi-LTSP 是一个替代项目；采用 SPICE 协议建设的云端复杂度较高，虽然可以全部采用开源软件，但必须具备一支技术水平相当高的建设团队；而采用 HDX 和 PCoIP 协议组建的云计算系统架构复杂，管理也复杂。

全部使用微软产品的应用环境使用 RDP 协议较合适；需要支持多种桌面操作系统的企业会发现 Citrix 的产品更加适合；如果希望通过使用瘦客户端或零客户端设备作为 VDI 解决方案的一部分，那么选择 PCoIP 技术可能更适合；局域网内的富图形处理业务选择 RGS 较明智。

总之，技术变化很快，各种协议都处于快速的发展变化之中，现在比较的结果与数月后的比较结果可能会有所不同，而且还可能有新的协议不断诞生。具备一定技术实力的公司可以在开源协议（SPICE、X、RFB）方面深入发展，并制定全部采用开源软件的云计算方案，这应该具备良好的发展前景。

第 6 章

“云”安全

面对扑面而来的云计算，人们既感到兴奋，又有些许忧虑，兴奋的是云计算能快速满足他们对计算资源的弹性需求，降低 IT 投入，以及能让他们摆脱复杂的 IT 技术而专注于自己的核心业务。然而，人们也非常关注云计算的安全风险，对于数据和应用放置别处而脱离自己的控制，他们心里没底。

最理想的信息安全可以描述为“我能自由地随时访问我的资料，且没经我授权的人无法获取我的信息”，可归结为使用自由、计算可用、数据不泄密，又可进一步细化为云用户是否自由、数据是否完整、数据是否泄密、数据是否一致、计算是否可用 5 个方面。传统的 IT 安全是通过采用在一定时期内数据不丢失、数据不泄密、数据没被破坏、计算持续可用的概率来衡量计算机安全性的高低，比如经常有人说“我们的系统安全性达到了 3 个‘9’”，意思就是能保证 99.9%的安全性，即一年内 99.9%的时间内系统是安全的。“云”安全也基本如此，是一个系统工程，不但涉及微观的技术层面，还涉及中观的提供商服务水平层面，以及宏观的国家政策法规层面。当然，云计算的类

型不同，在这三个层面的侧重也就有所不同，比如私有办公云侧重于微观和中观层面，而对于全国性的公有云安全来说，国家的政策法规不能缺位。由于用户的数据存在云端，因此用户自由度在云计算安全评价方面不可忽视。

6.1 信息管理与数据安全

信息安全的首要目标是保护我们的系统和应用所处理的数据资料。随着单位组织陆续把应用迁移到云端，甚至是迁移到几年前不可想象的外部或公有云端，传统的数据安全措施面临巨大挑战，随“云”而来的资源弹性、多租户、全新的物理和逻辑架构及抽象层控制，迫切寻求新的数据安全策略。

在云计算时代，如何安全地管理信息是所有组织不得不面临的一项艰巨任务，即使那些暂时还不用云计算的机构也不例外。管理信息具体包括内部数据管理、云迁移，以及被分散于多个单位组织的应用和服务中的数据的安全保障。信息管理和数据安全在云计算时代需要新的战略和技术架构。幸运的是，不仅用户拥有需要的工具和技术，而且迁移到云端后，数据还能得到更好的保护。

建议采用数据安全生命周期来评估和定义云端数据的安全策略，在制定明确的信息治理策略的基础上，通过诸如加密和专门的监测手段等关键技术来增强其安全性。

6.1.1 云信息架构

不同的云服务模式具备不同的云信息架构。无论是私有还是公共 IaaS 云服务模式，通常包括下面的存储。

- 原始块存储：存放数据的物理介质，如磁盘、光盘、磁带等。在一些特定的私有云中，这些物理介质能被直接访问。
- 卷存储：包括被附加到 IaaS 实例的卷（如虚拟硬盘驱动器）。在存储后

端，卷通常被打散存储以增强可靠性和安全性。卷不同于磁盘分区，磁盘分区是对一块基本的磁盘进行逻辑划分，比如从 101 柱面到 10 000 柱面划为一个分区，从 10 001 柱面至 50 000 柱面为另一个分区，因此一个分区的容量不可能超过磁盘的容量。而卷是在整合若干存储介质（如硬盘、分区、U 盘等）的基础上进行逻辑划分，因此一个卷允许跨越多个磁盘。

- 对象存储：通常指文件存储。不像虚拟机硬盘这种块设备，对象存储更像文件共享服务。
- 内容分发网络（CDN）：对象存储中的内容被分发到离用户最近的地方，以便增强终端用户的网络体验。

块存储、卷存储、对象存储和 CDN 的逻辑关系如图 6-1 所示。

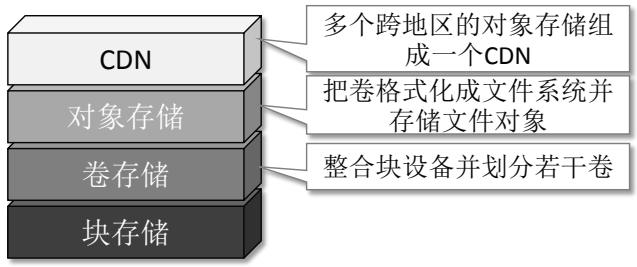


图 6-1　块存储、卷存储、对象存储和 CDN 的逻辑关系

PaaS 云服务在对外提供存储服务的同时也要消耗存储空间。PaaS 能提供的存储服务主要有数据库服务、Hadoop/MapReduce 大数据服务，以及应用存储（被整合到 PaaS 应用平台且通过 API 访问的其他存储）服务。而 PaaS 消耗的存储空间包括：

- 数据库。信息内容被直接存储在数据库（作为文本或二进制对象）或者被数据库表字段所引用的文件中，而数据库本身可能是共享后端存储的 IaaS 实例集合。
- 对象/文件存储。只能通过 PaaS API 访问。
- 卷存储。数据被存储在附加给 IaaS 实例的卷中，但这些数据专门供应

PaaS 云服务。

- 其他存储。不属于上述三类的其他存储。

类似于 PaaS 云服务，SaaS 云服务也提供存储服务和消耗各种存储资源，SaaS 云服务提供的存储服务一般通过基于 Web 的用户接口或者 C/S 模式的客户端访问，这点与通过 API 访问的 PaaS 云服务的存储服务不同，不过很多 SaaS 云服务提供商也对外提供 PaaS API 访问接口。

SaaS 云服务可能提供的存储服务包括：

- 信息存储和管理。比如现在流行的网络硬盘就是这类存储。
- 内容/文件存储。专门存储基于文件的内容，如网络相册等。

而 SaaS 云服务消耗的存储资源包括：

- 数据库。类似于 PaaS 云服务，绝大多数 SaaS 应用程序要依赖后端的数据库服务，甚至是文件存储服务。
- 对象/文件存储。文件和数据被保存在对象存储中，专门供应 SaaS 应用程序。
- 卷存储。数据被存储于附加在 IaaS 实例的卷中，并且专门供应 SaaS 应用程序。

云信息架构如图 6-2 所示。

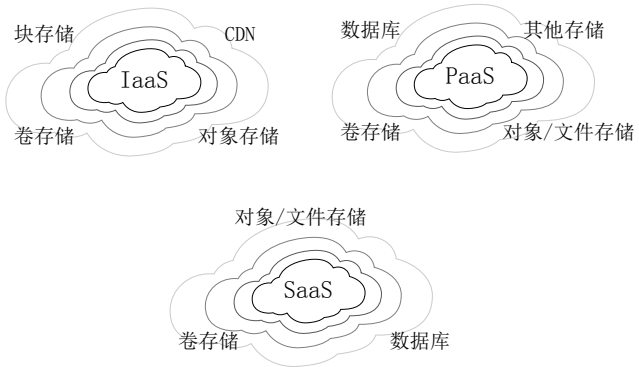


图 6-2　云信息架构

6.1.2 数据打散存储

数据打散存储是一种增强数据安全性的技术，它与加密技术不同，通过对数据分片，每个分片以多个副本的形式分散存储在不同的服务器上，以冗余存储换取数据的高可用性和高可靠性。目前绝大多数云服务提供商都采用了这种方法。例如，一个 500MB 的文件被划分为 5 个片，每片 3 个副本，一共 15 个片，被分散存储在多台服务器上，这样即使部分服务器损坏，文件也仍然不会遭到破坏。当用户读文件时，读取 5 个分片重新“组装”为一个完整的文件。数据打散存储结合加密技术，将会使数据的安全性得到进一步提高。

6.1.3 数据安全生命周期

尽管信息生命周期管理是一个相当成熟的领域，但是它并不能完全满足安全专家的要求，为此，人们提出了数据安全生命周期的概念。数据安全生命周期从创建到销毁共六个阶段，如图 6-3 所示。一旦被创建，数据可处于任何一个阶段，也允许跨到任何一个阶段，也可能不经过全部的六个阶段（比如，并不是所有的数据最后都要销毁）。准确跟踪数据处于数据安全生命周期的哪个阶段，是进行敏感数据保护的前提，同时能帮助确定在哪里应用安全控制措施。

图 6-3　数据安全生命周期

（1）创建：在第一阶段，人们创建结构化或非结构化的数据，如微软办公电子文档、PDF 文件、电子邮件、数据库中的记录或者图片文件。通常在此阶段，根据企业的数据安全策略对新产生的数据进行加密等级分类。

（2）存储：一旦创建了一个文件，它就被保存在某个地方。此时，你要确保存储的数据受到保护，同时应用了必要的数据安全控制措施。通过有效保护敏感数据，可以减少信息泄露的风险。本阶段通常与创建动作几乎同时发生。

（3）使用：一旦一个文件被创建并存储，那么随后可能将被使用。在这个阶段，数据被查看、处理、修改并保存。此时，在使用数据的过程中需要施加安全控制——你要能够监控用户活动并应用安全控制措施，以确保数据不被泄露。

（4）共享：数据经常在员工、客户和合作伙伴之间共享，因此必须要持续监控存储中的敏感数据信息。数据在各种公共的和私有的存储、应用程序和操作环境之间移动，并且被各个数据所有者通过不同的设备访问，这些情况可能会发生在数据安全生命周期的任何一个阶段，这就是为什么要在正确的时间引用正确的安全控制的真正原因。

（5）归档：数据离开生产活动领域并进入长期离线存储状态。

（6）销毁：采用物理或者数字手段永久销毁数据，物理手段如硬盘消磁，数字手段如加密切碎。

6.1.4 数据安全

数据是否完整、数据是否泄密、数据是否一致都属于数据是否安全的范畴。数据不完整是指在违背数据主人意愿的前提下，数据全部丢失或者部分丢失，数据所有者主动删除数据不属于数据不完整。数据泄密是指他人违背数据所有者的意愿而从数据中获取信息。下面几种情况不属于数据泄密：

（1）从网上下载的免费的并保存在计算机中的电影被他人复制了，因为

你不是电影的所有者。

（2）用 AES 加密过的一份个人文档被他人复制了，因为他人无法解密，从而无法获取里面的信息。

（3）一个没有加密的保存重要文档的 U 盘掉到大海里了，他人获得 U 盘中信息的概率可忽略不计。

（4）网上银行的密钥卡丢了。

（5）我主动把一份重要的方案材料传给客户。

上面的例子中，（2）表明他人虽然复制了经过 AES 加密的文档，但如果没有密码，他人是无法解密的，因为就算是使用当今最快的计算机进行暴力破解，也要花上一百多年的时间。（3）表明掉到大海里的 U 盘算是彻底损毁了，谁也得不到它，更不用说获取里面的信息了。在（4）中，网上银行的密钥卡丢了，就算别人捡到了，也无法操纵我的账户，因为他不知道我的网银账户和登录密码。其他两个例子更容易理解。

数据一致性是指数据没有错乱，能从中获取到这些数据所蕴含的全部信息。为了理解数据完整性和一致性的区别，请看下面的例子，如图 6-4 所示。

图 6-4　完整性和一致性

如果把数据比作毛线，那么图 6-4 左侧的毛线就表示数据是完整的而且是一致的；中间的被使用过的毛线团表示数据是不完整的但是是一致的；而右侧乱糟糟的毛线就表示数据是不一致的，很难从中抽出一根完整的毛线来，比如明显看到文件在硬盘里，但就是打不开，或者打开后显示乱码，这就是数据的不一致。

那么，数据存放在云端和存放在本地到底哪个更安全呢？结论如下：

数据存放在云端比存放在本地更安全。

原因如下。

1．数据完整性方面

云端通过采用服务器集群、异地容灾和容错等技术，可保证数据万无一失，采用数据快照回滚技术，能最大程度降低用户误删数据的损失，所以云端的数据丢失的概率极低；相反，如果数据保存在本地（计算机硬盘、U 盘、光盘、SD 卡、磁带等），这些存储介质都很容易损坏，另外没有任何措施可防止用户误删数据，现在的数据恢复公司业务火爆就充分说明了本地数据丢失的普遍性。

2．数据泄密方面

使用密码是目前最常用的防止数据泄密的方法，无论是云计算，还是使用本地计算机，都是如此。比如打开计算机，输入账号和密码登录，然后再输入密码登录 QQ、输入密码登录微博、输入密码登录邮箱、输入密码登录云等。另外，也有采用密码加密文档的，如密码保护的 Word 文档、压缩包等。

在当下云计算还不普遍的情况下，因为本地的存储介质（如硬盘、U 盘、SD 卡、光盘等）丢失而导致数据泄密的概率占到 70%以上，而其他诸如通过网络泄密的概率不到 30%。因此，把数据保存在云端，可以消除因丢失存储介质而泄密的可能性。另外，就算不用云计算，也存在网络泄密的可能性，除非你的计算机不连接网络。云服务提供商会采取各种防范网络泄密的措施，如防火墙过滤、入侵检测、用户行为异常分析、泄密预测等高精尖技术，个人用户是不可能花费巨资购买这些设备和技术的。最后，对于一些敏感的数据资料，用户如果实在不放心，还可以先加密，然后再保存到云端，常用的加密工具有 VeraCrypt、AxCrypt、BitLocker、7-Zip 等，也可以对 IaaS 存储产品（如虚拟机硬盘）全部进行加密处理。

3．数据一致性方面

数据没有错乱，没有遭到破坏，能正常打开和使用，这一点很关键。用过计算机的人应该都有过这样的经历：不正常关机（如突然停电、不小心按下计算机的电源开关或复位开关等）后重新启动计算机，报告硬盘文件遭到破坏需要修复，好不容易修复并启动完毕，发现之前辛苦几天编辑的 Word 文档打不开了，这就是各种干扰因素破坏了数据的一致性。存放在云端的数据一致性遭到破坏的概率要远远小于存放在本地计算机，原因很简单，云端环境更可靠：机房恒温恒湿、多级电力保障、阵列存储系统、异地灾难备份中心、安全防范措施全面、计算机专业人员维护等，这些措施使得数据不一致的概率几乎为零。

6.2　计算可用性

前面我们讲过 IT 的概念，即信息技术，其中“I”代表信息（或数据），“T”代表技术（或计算），技术是用来处理信息的，所以说 I 是目的，T 是手段，T 是为 I 服务的。上一节“信息管理与数据安全”关注的是 I，本节我们再来看看 T，与 T 关联的安全主要是“计算可用性”，如果由于 T 的原因，人们无法处理 I，那么这种情况就称为计算不可用。

首先来看下面几个计算不可用的例子：

（1）停电，计算机无法开机，所有的计算机软件都无法使用。

（2）操作系统损坏，计算机启动失败，保存在 U 盘上的数据完好无损，但是无法浏览和编辑。

（3）别人传了一个 PPT 文件给我，但是计算机上没有安装 Office 办公软件，所以我无法打开这个 PPT 文件。

（4）网络断了，无法通过 QQ 与外界联系，也无法上网和收发邮件。

（5）音箱坏了，而且没有耳机，所以无法使用计算机听音乐。

（6）我忘记了公司 ERP 的登录密码，所以无法使用财务系统。

计算不可用会导致人们无法处理全部或者部分数据，而断电、断网、软/硬件故障、缺少应用软件、忘记账号或密码等都会导致全部或部分计算不可用。相对于传统 IT 系统而言，云计算的确增加了一个可能产生故障的环节——云端，而且一旦云端崩溃，影响范围会很广，但不能因此而全盘否定云计算，因为世上根本就没有完美无缺的事物。一个公安局局长在否决私有云建设方案时说的一句话："万一云端出问题，大家都不用办公了？！"从正面回答公安局局长的问题，估计效果不大，可以顺着其思路提出如下类似的问题：

（1）万一银行的数据中心出问题，大家都不用取钱了？！所以银行不应该建立数据中心，但目前银行都建立了数据中心并实现了数据大集中。

（2）万一电厂出问题，所有的电器设备都不能用了？！所以家家都要自备发电机发电，但今天谁会自备发电机呢？

（3）万一移动公司的数据中心出问题，大家都不用手机了？！但目前几乎人人都在用手机。

（4）万一自来水厂出问题，大家都不用水了？！尽管存在这种断水的可能性，但哪家还会自挖水井呢？

（5）万一地球发生爆炸，大家都不用活命了？！所以人根本不应该活着，而实际情况是活着的我们有谁总担心地球发生爆炸呢？

类似的问题还有很多，不再一一列出。笔者从 1994 年—2002 年在银行工作，从事计算机技术岗位，见证并参与了银行从完全手工办理业务（算盘、手写存折）到全国数据大集中并实现网上银行的整个发展过程。1996 年以前，人们只能到开户行去存取款；而今天，我们不但能异地跨行存取款，而且坐在家里通过网上银行就能轻松实现支付和转账。总之，银行数据大集中利大于弊。

为了确保银行数据中心的可靠性，每家银行一般会在不同城市（如北京、上海和西安）各建一个中心，平时一个中心工作，另外两个作为灾备中心，三个中心的数据实时同步更新。这样，就算战争、地震等损毁了两个中心，也不会影响银行业务的正常办理。即使三个中心全部遭到破坏，也还有离线备份的数据，数据安全性绝对没有问题。

决策者和云计算工程师们会在计算的重要性和成本之间综合权衡，以建设一个令各方都满意的云端。例如，一个全国性的关系到每个人利益的公有云，就会考虑在不同的城市建立灾备云端，在另一个国家建立数据备份中心；一个涉及全省的公有云，可能会考虑在其他省的城市建立数据备份中心；一个世界 500 强企业的私有云，可能会建立一个异地灾备云端；一家中型企业的私有云，可能会引入集群、容错和故障转移技术；即使一家只有十几个人的小型公司的私有办公云，也会采用双机容错技术，只有当两台服务器同时出现故障时才会影响办公，但同时出现故障的概率几乎可以忽略不计，除非发生断电或地震等不可控事件。

就像银行数据中心的演化过程，IT 系统也遵循“手工→单机→私有云→公有云”的发展过程，每一次演化带来的好处要远远大于引入的弊端。

即使没有使用云计算的企业，在其 IT 系统中，也存在不少“单点故障”——如果出问题就会影响整个企业的业务办理。比如企业的邮件系统、门户网站、ERP 系统、文件服务器、局域网接入认证系统等，即 C/S 或 B/S 的 S 端（服务器端）就是单点故障点。

导致计算不可用的主要原因有停电、断网、硬件故障、软件故障（含病毒感染）。在传统 IT 系统中，个人计算机的软、硬件故障导致计算不可用的概率接近 95%，其中又以软件故障最为普遍，而因停电、断网导致计算不可用的概率微乎其微，服务器故障导致计算不可用的概率还不到 5%。而当一家企业采用云计算后，云终端是一个纯硬件、低功耗产品，而且 CPU、内存都焊死在主板上，又没有硬盘，所以云终端出故障的概率几乎可以忽略不计。云端采用多路供电、恒温恒湿系统，引入集群技术、容错技术及负载均衡技

术等措施确保计算持续可用，这样由云端、网络、终端组成的云计算系统具备极高的可靠性和安全性，计算可用性非常高。

6.3 互操作性与可移植性

不同于传统的位于单位组织内部的 IT 基础设施，云计算的诞生给单位组织的 IT 资源供给带来了前所未有的灵活性——可以瞬时增加、转移或者减少计算资源来快速响应动态的资源需求变化，能在几个小时内而不是几周内部署一个新的应用软件来满足业务需求。为了达到这种更具弹性的计算能力，在设计任何云系统时必须要考虑互操作性与可移植性。互操作性规定，IT 系统中应用软件层以下各层采用开放的通用型组件，杜绝使用云服务提供商各自内部的封闭组件；而可移植性规定，IT 系统中应用软件层和数据信息层应采用开放的通用数据格式和软件运行环境，保证同一个应用和数据能随意迁移到其他地方。一方面，互操作性与可移植性能让你把服务扩充到多个由不同云服务提供商提供的云端，在操作层面就像一个系统；另一方面，互操作性与可移植性能在不同平台或者不同云端之间轻松移动数据和应用。

互操作性与可移植性不是随云计算出现而产生的新概念，也不只是在云计算环境中人们才考虑互操作性与可移植性，只不过与传统 IT 系统相比，具备开放和共享处理能力的云计算更需要考虑互操作性与可移植性。多租户意味着多个单位组织的数据和应用并存，并且不排除能通过共享平台、共享存储和共享网络访问（有意或无意）他人机密数据的可能性。

下面重点介绍在设计互操作性与可移植性时必须考虑的关键因素。

6.3.1 互操作性

互操作性是云计算生态系统中各个协同工作的组件应具备的特征，这些组件可能来自各种云端和传统 IT 系统。互操作性使得我们可以随时使用新的

或者来自不同云服务提供商的组件来替换已有的组件，而不会中断云中的任务，也不影响数据在不同系统之间进行交换。

单位组织在使用云服务的过程中可能会考虑更换云服务提供商，常见的原因如下：

（1）续签合同时，云服务提供商的报价令消费者难以接受。

（2）消费者发现有价格更便宜的同类型云服务产品。

（3）云服务提供商停止了业务运营。

（4）云服务提供商在没有给出合理的数据迁移计划之前，关停了企业正在使用的服务。

（5）云服务提供商的服务质量难以令消费者满意，如未能达到服务水平协议（SLA）中规定的一些关键性能指标。

（6）云服务供/需双方之间存在商业纠纷。

如果缺少互操作性与可移植性，就会出现消费者被云服务提供商捆绑的现象，最终会损害消费者的利益。

一个云端互操作性的优劣程度往往取决于该云服务提供商是否使用开放的或者公开发布的架构和标准协议及标准的 API 接口。许多云服务提供商（如 Eucalyptus）喜欢在标准组件的基础上添加非公开的钩子（Hook）和扩展功能，以及一些增强功能，显然这些都会降低互操作性与可移植性。

6.3.2 可移植性

可移植性是指能把应用和数据迁移到其他地方而不用理会云服务提供商、平台、操作系统、基础设施、地点、存储、数据格式或者 API 接口。

在选择云服务提供商时，可移植性是需要考虑的一个重要方面，因为可移植性既能防止云服务提供商锁定用户，又允许用户把相同的云应用部署到

不同的云服务提供商那里，如建设灾备中心、同一应用全球分布式部署等。

如果未能妥善解决云迁移中的可移植性与互操作性，那么可能会无法实现迁移到云计算后的预期效益，并可能导致成本上升或项目延期，这是因为本该避免而未能避免的如下因素：

（1）云服务代理商或提供商锁定——一个特定的云解决方案的选择可能会限制以后转移到另一个云服务提供商。

（2）处理不兼容和冲突造成服务中断——云服务提供商、云平台和应用系统的差异可能会引发不兼容性，这种不兼容性会导致应用系统在不同的云基础架构中发生不可预料的故障。

（3）不可预料的应用系统重新设计或者业务流程更改——当迁移到一个新的云服务提供商时，可能需要重新审视程序的功能或者要求修改代码，以确保其最初的执行能力。

（4）高昂的数据迁移或数据转换成本——由于缺乏互操作性与可移植性，当迁移到新的云服务提供商时，可能会导致计划外的数据变化。

（5）数据或应用程序安全的丧失——不同的云服务提供商可能采用不同的安全控制、密钥管理或者数据保护策略，当迁移到一个新的云服务提供商或云平台时，可能会暴露原先未被发现的安全漏洞。

把应用迁移到云端是外包的一种形式，而外包的金科玉律是“了解前期并做好如何退出合同的计划”。因此，可移植性和在一定程度上互操作性应该是任何单位组织计划迁入云端时必须考虑的关键标准，同时制定好一个切实可行的退出方案。

6.3.3 实施建议

1．关于互操作性方面的建议

物理计算设备：同一个云服务提供商在不同时期的硬件或者同一时期不

同云服务提供商的硬件差别很大，如果直接让消费者访问硬件设备（如物理服务器），则会存在巨大的互操作性鸿沟。建议：（1）尽可能采用虚拟化来屏蔽底层硬件的差异，但是要注意虚拟化并不能屏蔽全部的硬件差异，特别是目前的系统。（2）如果消费者必须要直接访问硬件，那么从一个云服务提供商迁移到另一个云服务提供商时，选择相同或更好的物理硬件和管理安全控制就显得至关重要。

物理网络设备：不同的云服务提供商使用的网络设备（包括安全设备）也不尽相同，包括它们的 API 和配置流程也不尽相同。为了保证互操作性，网络硬件设备应该虚拟化，并尽可能使得 API 具备相同的功能。

虚拟化：虽然虚拟化有助于消除物理硬件的差异，但是常用的虚拟机管理程序（如 Xen、VMware、KVM 等）之间存在明显的差异。建议：（1）利用开放虚拟化文件格式，如 OVF，以确保互操作性。（2）了解并记录使用了哪些特定的虚拟化扩展钩子（Hook），虽然这些钩子与虚拟化文件格式无关，但是仍然存在这种可能性：在其他虚拟机管理程序中不起作用。

框架：不同的平台提供商提供了不同的云应用程序框架，而且这些框架之间确实存在影响互操作性的差异。建议：（1）在迁移到一个新的云服务提供商时，先搞清楚 API 的不同之处，然后拟定修改应用系统的计划。（2）使用公开发布的 API 以保证各组件之间具备最佳的互操作性，为了便于迁移应用和数据，有时很有必要变更云服务提供商。（3）云端的应用系统一般通过因特网交互，可能随时发生各种故障（如组件运行失败、网络中断等）。我们要做的是，确定一个组件发生故障（或反应慢）将如何影响其他组件，而且当一个远程组件出现故障时，要尽量避免可能会破坏系统的数据完整性的状态依赖——依赖另一个组件对数据的修改。

存储：数据类型不同对存储的要求也不同，结构化数据通常保存在关系数据库系统中，非结构化数据通常是按照应用程序（如微软的 Word 文字处理程序、Excel 表格程序和 Powerpoint 程序）所要求的格式存放。为了无缝地在不同云服务提供商之间移动数据，建议如下：（1）以一个被广泛接受的可移

植的格式存放非结构化数据。（2）评估数据在传输过程中是否需要加密。（3）检查各种兼容的数据库系统，如果需要，再评估不同数据库之间进行数据移植的难易程度。

2．关于可移植性方面的建议

把应用迁入云端的过程中会遇到很多问题，在可移植性方面的建议如下。

服务水平：不同云服务提供商的 SLA 也不同，所以有必要了解清楚 SLA 会如何影响将来更换云服务提供商。

不同的架构：云中的应用系统可以驻留在不同的平台架构中。通过了解应用与平台的依赖性，我们就能搞清楚这些（包括 API、虚拟机管理程序、应用程序逻辑等）将如何影响可移植性。

安全集成：在维护安全性方面，云系统引入了独特的可移植性担忧，具体包括：（1）现在云系统中的所有组件都会使用认证和身份机制，使用诸如 SAML 这种开放标准的身份机制将有助于提高可移植性，开发遵循 SAML 协议的内部 IAM 系统有助于系统将来移植到云中。（2）加密密钥应该托管在本地，并尽可能地在本地维护。（3）元数据是数据信息的一部分，但经常容易被忽视，因为我们在操作文件和文档时不能直接看到元数据。在云端，我们必须重视元数据，因为元数据随着文件一起移动，当移动文件和文件的元数据到一个新的云端时，要确保安全地删除了源文件元数据的全部副本，否则遗留下来的元数据可能会成为一个安全隐患。

6.3.4 用户自由度

互操作性与可移植性都是为了保障用户能自由使用云服务，这里的“自由”体现在以下几方面。

（1）用户可以自由地迁入、迁出云计算，不花或者花费很少的时间和金钱成本。

（2）用户能自由访问其他云应用：每个云端都是开放的，在这个云端中的用户能访问其他云端中的应用和资料。坚决杜绝运营商出于利益的考量封闭自身的云应用来黏住用户。

（3）用户能自由选择云服务提供商：买方能自由选择卖方是完全竞争市场的一个重要特征。为了保证自由度，政府应该努力营造完全自由竞争的云计算市场，坚决打击运营商“捆绑”用户的不正当竞争行为。

（4）用户能自由地把应用和数据从一个云服务提供商迁移到另一个云服务提供商：云端保存了用户的资料并安装用户需要的软件，当用户想更换云服务提供商时不应该存在障碍。扫除人为的障碍，杜绝不兼容性。所以，政府应该制定合理的云计算标准，并提供简单易用的迁移工具。

（5）云服务提供商能自由选择云组件：云服务提供商搭建云端，涉及很多技术和软/硬件产品，规划之初就要充分考虑伸缩性、开放性、标准性，杜绝被组件产品厂商捆绑。现在的很多云服务提供商热衷于采用开源的软/硬件产品，这是对的。

（6）云终端可以自由接入任何云端：一台终端设备成为人们的云计算总出入口。

6.4 加密与密钥管理

“如果你不信任他们，你就把数据加密。”这是数据安全专家常常挂在嘴边的一句话。部署在企业内部的数据中心，由于单位组织完全控制全部的软/硬件设备，所以数据要不要加密由企业规章制度界定。但是在云端，多个互不认识的租户共用计算资源，每个租户需要加密的数据自然就更多，需要加密的数据越多，业务流程越复杂，密钥管理工作也就越繁重。所以，在实际的操作过程中，我们要综合考虑各个因素，制定出一套既满足安全要求又不大幅度增加加/解密和密钥管理的工作难度的方案。

6.4.1 加密介绍

根据企业的规章制度，某些数据被列为机密并必须要加以保护，当前存储在企业内部的机密数据开始被陆续迁入云端，所以数据保护的力度必须要加大。把机密数据存储在企业的安全边界之外，这增加了数据保护的复杂度，同时也增加了风险系数。

关于云端的数据加密，以下因素必须慎重考虑。

（1）不单单在传输过程中需要加密保护，在云端存储和使用数据的过程中也依然需要加密保护。

（2）存储在云端并被共享的非结构化数据文件保护方法有两种：一是采用授权服务器进行集中加密；二是加密直接被嵌入到每个文件当中（分散加密）。当使用经第一种方法加密的文件时，首先要联系授权服务器进行解密，比如微软的 Word 文字处理软件会自动联系服务器并完成身份验证和解密工作。

（3）加/解密的密钥的管理期限就是数据的整个生命周期，在数据被销毁之前如果密钥丢失，则会导致数据无法解密。另外，对于密钥的保护和正当使用，要尽可能避免依赖云服务提供商。

（4）保护那些经常被忽视却又包含敏感信息的文件，比如日志文件和元数据如果不加以保护，那么很可能成为数据泄露的途径。

（5）云端的加密密钥应具备足够的强度（如 AES-256），且与同一单位组织内部的加密密钥一致。提倡使用开放的且经过验证的加密格式，尽可能避免使用特有的加密格式。

6.4.2 云端数据库加密

首先考虑数据要不要加密，因为加密数据会提高成本并使得业务处理复杂化。目前几乎所有的数据库都具备权限管理机制，如果配置得当，数据库

管理系统本身就能很好地保护敏感数据。如果遇到下面 3 种情况，那么就不得不对数据库中的记录进行加密了：

（1）不想让有权限的人（如数据库管理员）看到记录信息。

（2）法律规定一些记录必须要加密（如交易记录中的信用卡信息）。

（3）数据存储在数据库的一个 Schema 中，但是数据的主人无法控制访问这些数据的账号（比如多人使用的共享账号，每个人都不能随意修改账号和密码）。

当人们使用云端数据库，尤其是 SaaS 应用（该应用需要使用数据库）时，如果数据库不能操作加密的数据，那么数据库的功能会大打折扣。当然，前提是数据库管理系统或者应用程序能够访问密钥。

数据加密是以提高复杂度和降低性能为代价的，下面是一些替代的方法。

（1）使用数据库本身的对象安全机制：数据库中的表、视图、存储过程、函数等统称为对象，对这些对象的访问，数据库管理系统本身提供了一套权限管理机制，诸如账号、分组、角色、授权、撤权等。要严格控制那些被授权的账号，确保这些账号只分配给正确的人。

（2）存储安全散列值：只存储数据的散列值，而不直接存储数据本身。这使得应用程序能证明持有正确散列值的人就是持有正确数据的人，因为数据与散列值是一一对应关系，即 hash(data)=*x*，把 *x* 存储在数据库中，而 data 存储在另外一个私密的地方，从 *x* 是无法反算出 data 的。

6.4.3 密钥管理

无论是采用对称加密算法还是采用非对称加密算法，密钥的管理都是重中之重，尤其是对于多租户模式的公有云端来说，密钥管理是一个比较繁重的任务。

最简单的案例是应用软件在云端运行，而只在企业内部使用密钥对迁移

到云端的数据进行加密——在企业的网络安全边界部署一个加密引擎，只要通过这个加密引擎，那么离开企业的数据就会自动加密，而进入企业的数据则会自动解密。而下面这种情况会把事情搞得很复杂：如果一个使用密钥的应用软件又包含其他需要使用密钥解密数据的进程（如一个批处理应用软件可能包含很多个进程），且这些进程又驻留云端，那么此时的密钥需要离开企业内部的网络安全边界而进入云端。

企业一般为每个需要加密功能的实体（用户、设备、进程等）分配单独的密钥，这样就不用共享一个密钥，从而避免带来很多隐患。为了管理众多的实体密钥，最简便的方法是部署一个基于身份的密钥管理中心，从而使得为一个具体实体加密的任何数据只对该实体本身有效。如果同一个组的成员确实需要共享数据，那么可以给该组分配组级的密钥，同组成员共享组级密钥。正如前面所提到的那样：密钥的管理必须局限在企业内部。

如果数据存储在公有云环境中，那么在退出这个环境时，需要确保所有的数据（特别是 PII、SPI 数据或受到监管的数据）已经从公有云环境中删除，其中包括其他存储介质（如备份磁带）上的数据。本地管理密钥的做法很容易实现这个目标：只要从本地密钥管理中心删除相应的密钥即可，从而保证残留在公有云的任何数据再也不能被解密。

如果云服务提供商和消费者不严格执行各自的密钥管理流程，那么数据加密就没有什么实际意义。比如云服务提供商，如果职责混乱，大家可以随意访问密钥服务器和存储了加密数据的服务器，或者 DBA 能随意访问数据库中的个人密钥，那么数据加密就形同虚设。同样，比如云服务消费者，如果存储密钥的终端设备本身就不安全（如移动设备）或者没有达到与加密系统同等安全级别的终端设备，这就像用木板做保险柜，那么这时加密数据无疑也是形同虚设。

围绕如何保护密钥本身，方案设计师通常的做法是采用密钥来加密密钥，只在内存中产生有效密钥，并且只存储加密过的密钥。这理解起来有点绕，其实就是把密钥当作普通的数据再次进行加密。

6.5 云服务提供商

前面讲到，经营云端的公司应具备更多的投入和技术以保证其安全性，但是一个真正落地的云端能不能达到理想的安全级别，还要看云端运营公司的声誉、品牌、技术实力，以及当地政府的监管力度。欧美国家的云服务提供商普遍好于国内的公司，它们资金雄厚、技术实力强、品牌声誉好。选择一家技术过硬、实力雄厚、信誉良好、服务到位的云服务提供商很重要，因为云服务不是一次性买卖，供需双方需要长久合作。

为了搞清楚一家云服务公司的产品线，我们首先要理解一个传统数据中心的组成，否则很难搞清楚一家云服务提供商提供的众多产品线的作用和它们之间的关系。传统的数据中心由若干个局域网、一定数量的服务器、若干存储设备、一些负载均衡器、入侵检测设备、域名服务、数据库服务、DNS 服务、DHCP 服务、邮箱系统、门户网站、目录服务、用户身份认证和权限管理服务、文件服务、虚拟办公桌面等组成。如果一家云服务公司能全面接管一家企业的数据中心，而这家企业只需摆放云终端和出口网络设备，那么这家云服务公司的产品线是最全面的。

截止到 2018 年，云服务提供商以美国企业为主。中国云服务提供商发展迅速，而且绝大多数的云服务提供商提供 IaaS 和 PaaS 类型的云服务，而紧贴人们生活的 SaaS 类云应用软件不多。下面简单介绍一下目前一些典型的云服务提供商。

1．亚马逊 AWS

亚马逊刚开始是做电商的，购买了一批服务器搭建电商平台，由于服务器具备富余的计算资源，于是考虑对外出租这些资源，从此开展了云计算业务并越做越大。现在，亚马逊算是世界上最大的云计算服务公司，产品线丰富，涵盖了 IT 系统架构的各个层次，加上另外几个部署和运维产品，一个企业的数据中心可以采用亚马逊云计算服务产品来完全替换。官网网站为 https://aws.amazon.com/。

2．微软 Azure

微软云端的技术绝大多数是自己的，如 Windows 操作系统、SQL Server 数据库、Office 办公软件、活动目录等。优点是架构简洁、综合成本低，但是互操作性和可移植性有待进一步提高，虽然目前已经引入了 Linux、Hadoop、Eclipse 等开源产品，但还不够。微软虽然诞生于传统“闭源”时代，但是能与时俱进以开放心态走进当今共享的互联时代，实属难能可贵。虽然微软的云计算业务起步较晚，但是今日它的云计算产品线极其丰富，成了亚马逊的主要竞争对手，且这块收入也占据公司总收入的半壁江山。这点值得华为学习和借鉴，华为在开放方面应该更大胆一些。微软的云端分布于全球各地，不过官网对云产品的介绍有点凌乱，微软云官方网站为 https://azure.microsoft.com/。

3．谷歌云平台

2017 年谷歌公司把之前的各种零星的云产品整合，正式推出云平台。谷歌的云计算服务产品线非常丰富，而且在大数据和人工智能方面存在优势。谷歌云平台的安全性是有目共睹的，服务器、网络设备都自己定制，而且开发了一系列的安全芯片，确保从硬件到软件，从启动到运行全方位安全保障。凭借自己的雄厚技术和资金实力，可以肯定谷歌在云计算市场会异军突起，很快就会与亚马逊和微软形成三足鼎立之势。但是截至 2018 年 8 月，谷歌云计算业务还未能进入中国，不过谷歌正在努力契合中国的法律法规，相信中国大陆的用户很快就能使用谷歌优秀的云服务产品了。谷歌公司的云端分布于全球，用户的关键数据可以做到以多副本方式保存在地球的各大洲，真正能做到地球在数据就在了。谷歌云平台的官方网站为 https://cloud.google.com/。

4．阿里云

阿里云拥有诸如虚拟主机、存储和虚拟网络等核心云服务产品，经过最近几年的快速发展，产品线越来越丰富，出口带宽和稳定性在国内的云服

务公司中相当不错。凭借国内的独特环境，阿里云已经成长为国内最大的云服务提供商。但是全球布局还任重道远，加上中国国情的特殊影响，跨国企业在短时间内很难把阿里云作为云服务产品的合作伙伴。阿里云官方网站为https://www.alibabacloud.com。

5．华为云

在 2017 年 6 月，本书出第二版时，华为的云服务产品还很少，一年后的今天华为推出的云服务产品数目和种类令人吃惊，几乎与亚马逊相当，不过在可靠性方面的保障需要投入巨大的时间和精力。虽然华为具备硬件上的技术优势和底蕴，几乎覆盖了产业链的“云、管、端”，但是在开放的互联网时代，这恰恰又是它的劣势，其对技术的保守和封闭性不契合于当今这个开放时代。加上国际环境大转向，这对华为等中资企业来说无疑是“恶劣的”。产品进不了欧美国家，就算超过了阿里，又能怎么样呢？只要国内环境依旧，在云计算领域，笔者对国内企业的发展前景不乐观。

6．提供商产品比较

时至今日，每家公司的云服务产品都非常多，产品名称相互存在差别，根据功能统计，结果见表 6-1，一些次要的非主流产品并没有列出来。

表 6-1　主要云服务提供商的产品比较

云产品 \ 提供商		亚马逊	微软	谷歌	阿里	华为
计算	虚拟主机	√	√	√	√	√
	VPS				√	
	物理主机				√	√
	容器	√	√	√	√	√
	无服务器计算(Serverless)	√	√	√	√	√
	加固虚拟机(Shielded VM)			√		

续表

云产品 \ 提供商		亚马逊	微软	谷歌	阿里	华为
存储	块存储	√	√	√		√
	文件存储	√	√	√	√	√
	对象存储	√	√	√	√	√
	备份	√			√	√
	CDN	√	√	√	√	√
	存档存储	√	√			
	数据迁移设备	√	√	√	√	√
网络	VPC	√	√	√	√	√
	专线	√	√	√	√	√
	VPN		√		√	√
	公网 IP	√	√	√	√	√
	负载均衡	√	√	√	√	√
	DNS 服务	√	√	√	√	√
数据库	SQL 关系数据库	√	√	√	√	√
	NoSQL 数据库	√	√	√	√	√
	分布式内存数据库	√	√	√	√	√
	时序数据库			√	√	
	数据仓库	√	√	√		√
	图数据库	√				√
AI 和机器学习	机器学习模型	√	√	√	√	√
	视频分析	√	√	√	√	
	图片分析	√	√	√	√	√
	自然语言理解	√	√	√		√
	语音识别	√	√	√		√
	人脸识别	√	√			√
	文本朗读（多国语音）	√		√		
	语音对话	√		√		
	深度学习	√		√		

续表

云产品 \ 提供商		亚马逊	微软	谷歌	阿里	华为
物联网	IoT 中心	√	√	√	√	√
	IoT 边缘	√	√	√		√
	小硬件设备	√		√		
安全	DDOS 保护	√	√	√	√	√
	应用防火墙	√	√	√	√	√
	IAM	√	√	√		√
	密钥管理	√	√	√	√	
	入侵检测	√	√	√		
应用服务	API 网关		√	√	√	√
	协作&生产力	√	√+	√+		√
	区块链					√
	微服务					√
	地图&位置服务			√+		
	云桌面	√+				√
媒体服务		√++	√	√		√+
数据迁移		√+	√	√		√+
运维	资源预置	√				
	配置管理	√			√	√
	监控	√	√	√	√	√
	管理与合规性	√	√			
	资源优化	√		√		
大数据	Hadoop 平台	√	√		√	√
	数据流处理	√	√	√		√
	ETL	√		√		
	BI	√	√	√	√	
	BigQuery			√		
中间件	消息服务	√			√	√
	SNS	√				

续表

云产品 \ 提供商		亚马逊	微软	谷歌	阿里	华为
移动应用		√++	√	√-		√-
云端分布（区域个数）	北美洲	20(33)	16	19	2	1
	南美洲	3(4)	1	3		4
	亚洲	17(24)	15	18	10	7
	欧洲	12(23)	14	18	1	2
	非洲	(2)	2			
	大洋洲	3(2)	4	3	1	

关于此表的一些说明如下。

（1）信息来源于各个云服务提供商的官方网站，对数据的准确性笔者不负责。

（2）“AI 和机器学习”大类评价的指标特别多，比如准确度、语言种类数量、响应时间、学习速度、抗干扰性等，尽管国内提供商官网也列举了一些产品，但是综合评价都较低，最好的是谷歌、然后依次为亚马逊、微软等。

（3）华为云端没有在美国、英国、日本、韩国和澳大利亚等发达国家设云端。谷歌没有在非洲、中国、俄罗斯、印度设云端。亚马逊云端分布标注的 $x(y)$中，x 表示可用区域个数，y 表示边缘网络站点个数。

（4）“√”号后有“+”号，表示相应的云服务产品包含更多的子项，“++”表示子项数目更多。

（5）很多云服务产品都是采用公开的技术和开源产品来实现的，所以云服务产品的类型和数目各个公司大致相同。亚马逊、谷歌和微软是开源技术的重要贡献者，所以这三家公司的云服务产品大部分是原生的。在选择云服务提供商时，提供商的技术实力和云端全球布局情况是两个非常重要的指标，让用户分布在云端 100 毫秒的时延半径之内，这很重要。

（6）非洲、俄罗斯几乎没有云端。

（7）国内云提供商受到政府严格管控，给租户带来不自由，比如需要备案，需要在虚拟机里安装监控程序等。

第 7 章

“云”应用

云应用不同于云产品，云产品一般是由软硬件厂商开发和生产出来的，而云应用是由云计算运营商提供的服务，这些运营商需要事先采用云产品搭建云计算中心，然后才能对外提供云计算服务。在云计算产业链上，云产品是云应用的上游产品。

云计算的目的是云应用，离开应用，搭建云计算中心没有任何意义。中国目前的云计算中心如雨后春笋般涌现，都是政府大手笔的投资，但是云应用却很少，所以有专家惊呼“云计算并不是绿色 IT，其能耗更高”。机器空转，没有产能输出，当然是无谓地浪费了更多的能耗。本章列举一些最典型的云应用例子，各个行业内部使用的“云”五花八门，这里不再一一列出。另外，我们要注意的是，云应用种类非常多，但是构成上都遵循相同的“云”、“管”、“端”的结构，在“端”的表现尽量要单一且标准化，典型的云终“端”就是在用户手持设备上安装一个 APP 而已。

云应用的第二个特点就是具备某种程度的智能性，在一定的基础数据上

进行大数据分析，从而表现一定程度的人文关怀，而不是冷冰冰的机器。我们只有俯视它，才能更好地理解各种云应用；如果仰视，你就落入世俗的套路了。

7.1 企业私有办公云

与传统的以计算机为主的办公环境相比，私有办公云具备更多的优势，比如：

（1）建设成本和使用成本低。

（2）维护更容易。

（3）云终端是纯硬件产品，可靠、稳定且折旧周期长。

（4）由于数据集中存放在云端，从而更容易保全企业的知识资产。

（5）能实现移动办公，员工能在任何一台云终端上使用自己的账号登录云端办公。

关于私有办公云的更多优势描述，请参考 2.8 节。

比如，一个小企业（员工人数少于 100 人），采用两台服务器做云端，办公软件安装在服务器上，数据资料也存放在服务器上。通过有线或无线网络连接到办公终端，给每个员工分配一个账号即可，员工随便在哪台终端都可以用自己的账号登录云端办公。企业私有办公云如图 7-1 所示。

在外出差的员工，可以通过 VPN 登录到公司内部的云端。

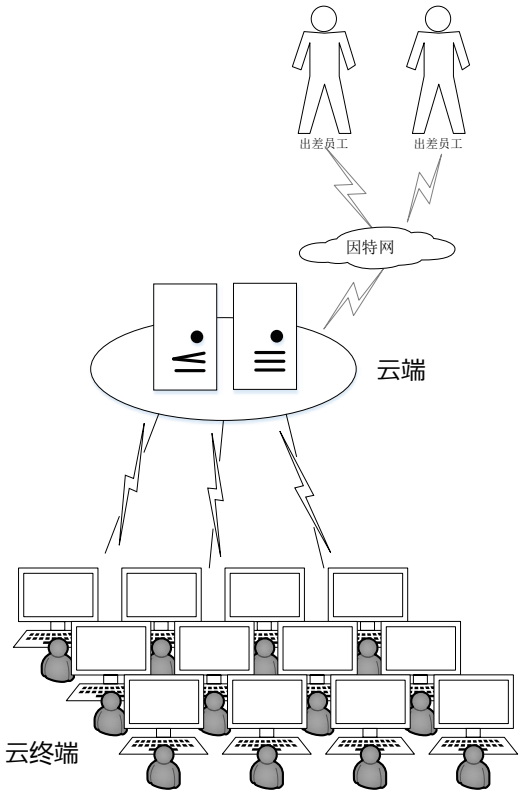


图 7-1　企业私有办公云

7.2　园区云

园区内的企业经营的产品具有竞争关系或者上下游关系，企业的市场营销和经营管理具有很大的共性，且企业相对集中，所以在园区内部最适合构建云计算平台。由园区管委会主导并运营云端，通过光纤接入区内各家企业，企业内部配备云终端。云端应该“飘着”这样几朵云：

（1）**企业应用云**。ERP（企业资源计划）、CRM（客户关系管理）、SCM（供应链管理）等企业应用软件是现代企业的必备软件，代表着企业研发、采购、生产、销售和管理的流程化和现代化。如果园区内每家企业单独购买这

些软件，则价格昂贵、实施困难、运维复杂、二次开发难度大，但经过云化后部署于云端，企业按需租用，价格低廉，则所有难题迎刃而解。

（2）**电子商务云**。为了覆盖尽量长的产业链条，引入电子商务云，一方面对内可以打通上下游企业的信息通路，整合产业链条上的相关资源，从而降低交易成本；另一方面对外形成统一的门户和宣传口径，避免内部恶意竞争，进而形成凝聚力一致对外，这对于营销网络建设、强化市场开拓、整体塑造园区品牌形象具有重大意义。

（3）**移动办公云**。在园区内部署移动办公云，使得园区内企业以低廉的价格便可达到如下目的：使用正版软件、企业知识资产得以保全、随时随地办公、企业 IT 投入大幅度下降、应用部署快速、从繁重的 IT 运维工作中解脱出来并专注于自己的核心业务。

（4）**数据存储云**。如果关键数据丢失，则 80%的企业要倒闭，这已经是业界的共识。在园区部署数据存储云（必要时建立异地灾备中心），以数据块或文件的形式通过在线或离线手段存储企业的各种加密或解密的业务数据，并建立数据回溯机制，可以规避如下事故导致的企业数据丢失或泄密风险：存储设备毁坏、计算机被盗、发生火灾、发生水灾、房屋倒塌、地震、战争、雷击、误删数据等。

（5）**高性能计算云**。新产品开发、场景模拟、工艺改进等往往涉及模拟实验、数学建模等需要大量计算的子项目，如果只靠单台计算机，则一次计算过程往往会耗费很长时间，而且失败率居高不下。因而，园区统一引入高性能计算云和 3D 打印设备，出租给需要的企业，从而加快产品迭代的步伐。

（6）**教育培训云**。抽取当前各个企业培训的共性部分，形成教育培训公有云平台，实现现场和远程培训相结合，一方面能最大限度地减少教育培训方面的重复建设，降低企业对新员工和新业务的培训投入，加强校企合作，集中优良师资和培训条件，使教育培训效果事半功倍；另一方面又能通过网络快速实现“送教下乡”。

构建园区云能够大幅度提升园区服务管理水平，积极影响潜在入园企业，提高入园企业满意度，促进孵化企业成长步伐，达到“企业进得来、留得住、发展快”的目的。

7.3 医疗云

医疗云的核心是以全民电子健康档案为基础，建立覆盖医疗卫生体系的信息共享平台，打破各个医疗机构信息孤岛现象，同时围绕居民的健康关怀提供统一的健康业务部署，建立远程医疗系统，尤其使得很多缺医少药的农村受惠，医疗云如图 7-2 所示。

图 7-2　医疗云

依托医疗云，可以在人口密集居住区增设各种体检自助终端，甚至可以使自助终端进入家庭。建立医疗云利国利民，其重大意义归纳如下。

（1）**对于国家公共卫生服务管理部门**：有利于公共卫生业务联动工作；有利于疾病预防与控制管理；有利于突发公共卫生事件处理；便于开展公共

卫生服务；有利于资源整合、减少重复投资，甚至可以把检查检验功能独立开来，专门成立第三方机构；便于实现跨业务、跨系统的数据共享利用。

（2）**对于医疗卫生服务机构**：有利于提高医疗服务的质量；有利于节省患者支出，缓解群众看病贵的问题；便于争抢生命绿色通道的“黄金时间”；有利于充分共享医疗资源；有利于开展远程医疗业务。

（3）**对于社区卫生服务站**：有利于开展“六位一体”业务；有利于开展健康干预跟踪服务。

（4）**对于个人**：能减少重复的检查检验开支；便于“移动”（如转院、跨地区等）治病；通过远程医疗系统便于享受优质的医疗服务；医疗云结合大数据能预测个人疾病，所以能提前预防重大疾病的发生。

7.4 公民档案云

我国基于纸质档案和户口的管理体系极其落后，消耗了大量的社会资源。经历过档案调动或户口迁移的人都苦不堪言，长途奔波，数月折腾，费时、费财、费精力。如果由中央政府牵头建立和运营全国性的公民档案云，全部纸质材料电子化后集中存放在云中，再辅以公民的指纹数据，并给每个公民发放一对独一无二的公/私钥，而且还可以纳入公民和企业的诚信信息、学历资料等。个人和企事业单位通过安装 APP 就能查阅授权的资料。这才是真正的功在当代、利在千秋的工程。有了这个公民档案云后，以下事情将变得异常简单。

（1）人口普查。通过简单查询即可按各种口径统计人口信息。

（2）档案调动。

（3）户口迁移。不过，户口迟早会取消。

（4）诚信信息检索。

（5）户口注册和注销。

（6）简历查阅。

（7）公共决策。通过大数据分析，可以为许多公共决策提供依据。

（8）电子材料数字签名。利用自己的私钥签名，接收方利用公钥验证，无人能冒充。

（9）血型、器官移植匹配。

7.5 卫生保健云

不同于医疗云，卫生保健云侧重于个人、家庭、家族的卫生、保健、饮食、作息等信息的收集、存储、加工、咨询及预测等，重在关怀国民的身体状况，覆盖从出生到死亡的全过程。建设主体也是中央政府，为国家层面的民生项目。鼓励企业开发各种体检和检验终端设备，如智能手环、家庭简易体检仪、小区自助体检亭、老人和小孩定位器、监护仪等。体检终端设备发放到千家万户，实时收集国民的身体状况数据，云端程序 7×24 小时监测这些数据，并及时把分析结果发送到公民的云终端设备上。当沉淀大量保健数据后，就可以采用大数据进行各种定性分析，如疾病预测、饮食建议、流行病预测控制等。卫生保健云可与医疗云、公民档案云建立联动。

7.6 教育云

构建教育云是一个庞大的系统工程，由一个国家层面的公共教育云和成千上万的学校私有教育云组成，而且私有教育云建设要先行启动，教育管理部门制定标准，由各个学校自己主导建设。公共教育云应该由中央政府牵头完成，承载共性教育资源和标杆教育资源，同时作为连接各个私有教育云的

纽带。各个学校的私有教育云承载各种特色资源，履行“教”与“学”的具体任务。每个学校运营自己的私有云端，而云终端发放到每个老师和学生的手上，形态上可以是固定云终端（放置在老师办公室、机房、多媒体教室、图书馆的多媒体阅览室等）、移动云终端（发给老师和学生）、移动固定两用云终端及多屏云终端。云端和云终端通过校园高速光纤互联在一起，教育云如图 7-3 所示。

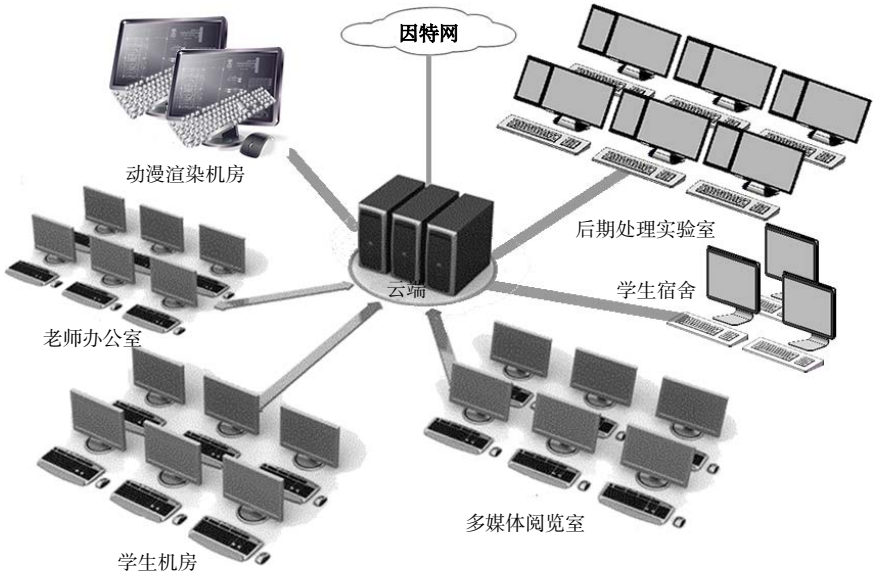


图 7-3　教育云

新生注册时，为每个学生分配一个云端账号和一台手持云终端，一个账号对应一个虚拟云桌面，学生毕业后回收其云端资源。在机房、宿舍、图书馆等场所，只要坐下来，就可以把手持设备插入固定云终端，然后就可以使用键盘和屏幕了。手持设备也可以单独接入云端。与传统的非教育云相比，学校采用私有教育云有如下好处。

（1）移动教学。无论师生在哪里，都能登录自己的云端桌面。

（2）延续实验。由于每个学生都有自己独有的虚拟机，所以跨节次的实验不会被中断。

（3）远程教学。老师能选择云端的任何学生的云桌面并广播课件。

（4）规范学生用机行为。能轻松控制学生可以安装和使用的软件，杜绝学生沉迷游戏。

（5）便于资源共享。

（6）便于学生积淀学习笔记和素材。

（7）便于计算机系学生云中开发。

（8）轻松实现高性能计算，如科学研究、动漫渲染、游戏开发、虚拟现实模拟等。

（9）便于因材施教。在掌握一定的基础知识后因材施教，最大限度地发挥每个学生的特长，这是最理想的教育方法。利用虚拟现实技术产生学生喜爱的“老师”，利用大数据分析为每个学生制订教学计划，然后给每个学生分配一个“老师”，按照制订的计划来一对一教学。

在私有教育云的基础上再抽取共性资源，形成全国性的公共教育云，同时引入虚拟现实技术，实现远程教育，使得偏远的广大农村受益。

7.7 交通云

交通云将车辆监控、路况监视、驾驶员行为习惯等错综复杂的信息，集中到云计算平台进行处理和分析，并推送到云终端。建立一套信息化、智能化、社会化的交通信息服务系统，使国家交通设施发挥最大效能。可以为每位驾驶员和每辆机动车建立档案，收集车辆位置、车况、车内空气、车辆保养、车辆维修、司机驾驶行为等信息。经过云计算处理后，一方面把结果（如交通路况、驾驶提醒、保养提醒等）反馈给司机和他的家人；另一方面利用大数据分析，预测车辆故障和交通事故的发生，提前做好预防措施，这将大大减少交通事故和人员伤亡。同时交警、汽车厂商、保险公司、维修部、汽

车俱乐部等部门通过交通云都能获取相应的信息。

7.8 出行云

出行云涵盖天气、地图、公共交通、景点、人文风俗、酒店、特产等信息资源，覆盖人们的旅游、度假、出差、探亲等活动。出行云应该算是 SaaS 公有云，通过安装 APP 呈现到人们的云终端设备上。出行云重在对出行在外的人施以关怀，而且建立与家人的多方式联系和互动，覆盖行前、行中、行后三个阶段。在积累一定量的数据之后，出行云运用大数据分析人们的喜好和行为习惯，在合理的时间向其推送合理的建议，使人们感觉到出行云是其导游、生活顾问、仆人、朋友。

7.9 购物云

购物的过程和目的都是体验，最理想的体验就是在正确的时间以合理的价格买到称心如意的商品且符合自己预期的使用目标。一次完整的购物消费过程包括 8 个阶段：产生需求→形成心理价位→选择商品→付钱购买→接收商品→使用商品→售后服务→用完回收。每个阶段都是一个选择、分享和评价的过程。购物云必须完全覆盖这 8 个阶段，且在每个阶段灵活引入相应的关怀和分享机制，比如：

（1）购物云咨询其他云（如公民档案云、卫生保健云等）科学预测用户的需求，并在合理的时间点提醒用户需要购买什么商品。

（2）咨询其他云，从而合理计算出用户购物的心理价位区间。

（3）选择商品时，用户只需采用自然语言说出需求信息，购物云就会返回满足需求的商品列表，并且通过虚拟现实技术给用户建模，让其“进入”云中体验商品，如试穿衣服、触摸家具等。现实中的人们观看云中的试用商

品的情景，也可以观看云中的“其他人”试用商品的情景，并且可以分享各自的观点，这比实体店购物体验更好。

（4）购物界面上始终呈现一个虚拟的购物顾问，它其实就是一个无所不知的购物机器人（类似微软的小冰），用户可以向它咨询任何问题。

（5）付钱购买直接在购物云中完成，无须登录网上银行。

（6）开辟高档商品俱乐部，线上、线下形成圈子，大家分享各自的商品使用体验。

（7）给每个注册的网购用户都提供一名虚拟的咨询顾问，对于购买的任何商品，虚拟的咨询顾问都会给用户无微不至的关怀，比如提醒保养，用户也可以随时向它询问。

总之，与传统的网店相比，购物云具备更好的智能，提供比线下购物更佳的用户体验，它其实就是你的亲密朋友。

7.10 农村农业云

城乡巨大的数字化鸿沟已经引起中国政府的高度重视，在信息化时代，没有网络的地方就是与世隔绝，也就是新的“井底之蛙”。信息不流畅必定会导致物流不流畅，物流不流畅的必然结果就是阻碍市场交换，影响百姓生活。这几年出现的“农民有菜卖不出、市民吃菜买不起”的现象就是活生生的教训，农民养的是本地鸡，下的可是全球蛋，一根网线通天下。农民对互联网不是没有需求，而是客观条件限制了他们的需求诉求：农民计算机知识匮乏、计算机售后服务缺位、环境恶劣（如电压不稳定、无地线、多雷击、气候潮湿、冬冷夏热、鼠害严重、卫生条件差等）等。

云计算是解决这些问题进而推进农村农业信息化建设的最佳手段——软件上移云端，很少出故障的纯硬件的云终端进入农村家庭，这样就能降低使

用难度、减少故障发生率。建设农村农业云，可概括为“三化”：软件云化、终端固化、操作傻瓜化。一个典型的云计算模型是在县级或者市级建设一个云中心，通过光缆连接到各行政村，再通过有线或无线方式连接到云终端。

把农村农业云的云端上交到县或市级政府，由专业技术人员管理，云端安装常用的软件，存放各种涉农教育音视频资料，并为每个用户开辟存储空间，用于存放用户自己的配置和信息数据。云中心建设严格遵循可伸缩性原则，由小变大，规模由用户数来驱动。

7.11 高性能计算云

高性能计算云，即把云端成千上万台服务器联合起来，组成高性能计算集群，承载中型、大型、特大型计算任务。比如：

（1）科学计算，解决科学研究和工程技术中所遇到的大规模数学计算问题，可广泛应用于数学、物理、天文、气象、化学、材料、生物、流体力学等学科领域。

（2）建模与仿真，包括自然界的生物建模和仿真、社会群体建模和仿真、进化建模和仿真等。

（3）工程模拟，如核爆炸模拟、风洞模拟、碰撞模拟等。

（4）图形渲染，应用领域有 3D 游戏、电影电视特效、动画制作、建筑设计、室内装潢等可视化设计。

7.12 人工智能云

以其他云为基础，诞生的人工智能云可以算是人类追求的终极目标，它具备浩如烟海的知识，具备人的智慧、人的情感和超强的运算速度，能学习、

能推理、能和人类进行语言互动，它还会做科学研究。人工智能云的触角深入到人类生活的方方面面（如果把各种传感终端当作触角），它改变并影响每个人的日常生活、学习和工作习惯——它监测每个人的身心健康、饮食习惯，并能作出疾病预测。人工智能云是全球性的公有云，每个国家都在为它贡献自己的力量，不断完善其算法，充实其知识，规范其行为。在人工智能云的笼罩下，地球真正变成了一个村子，人们交流无障碍，这里的人充满良善、爱心和慈悲心。

其他云成了人工智能云的数据来源，人工智能云成了人们唯一的交互云平台。比如，我们再也不用去购物云上购物了，因为人工智能云已经自动为我们购买了需要的东西；我们也不用关心出行云了，因为人工智能云已经为我们准备好了一切：行程安排、酒店预订、饮食准备等。自动驾驶的交通工具就是一台云终端设备，即人工智能云的触角。人工智能云能陪我们聊天、下棋，也能教小朋友知识，它成了各种机器人的超级大脑。家庭机器人就是人工智能云的云终端，机器人本身只是执行部件并做一些常规的判断，复杂的推理交给云来完成。计算能力超强的智能云能瞬间作出判断并给机器人反馈结果，因而这样的机器人表现极其聪明，反应敏捷，如果不好好控制，那么会给人类带来威胁。

第 8 章

“云”实验

本章采用 OpenStack 搭建一个具体的云计算环境，并感受管理一个云端的基本过程，以加深对前面章节内容的理解。

OpenStack 官方文档将节点分为四类，它们分别是控制节点（controller node）、网络节点（network node）、计算节点（compute node）和存储节点（storage node）。这里的“节点”（node）就是服务的集合，而服务就是一个对外提供功能的进程（即计算机里正在运行的程序）。这四类节点只是对 OpenStack 的所有进程及依赖的第三方进程做了分类，它们不会相互排斥，也就是说可以将四类节点安装在不同的服务器上，也可以安装在同一台服务器上，但要特别注意的是某个进程允许同时属于多个节点。在实际操作时，具体的服务器要安装什么进程，成为什么节点，完全取决于安装规模和实际的环境。由于本实验没涉及存储节点，所以表 8-1 没列出存储节点上的服务。

表 8-1　各个节点中包含的服务

节点名称	服　务	功　能	软　件
控制节点	Http Service	网站服务	Apache2
	SQL Database Service	存储结构化数据	MariaDB | MySQL | PostgreSQL
	Message Queue	转发消息	RabbitMQ | ZeroMQ
	Network Time Service	同步各台机器的时钟	chrony | ntp
	Memcached Service	在内存中缓存令牌	memcached, python-memcache
	Identity Service	认证客户端身份	keystone
	Image Service	管理虚拟机镜像文件，由第三方（如 Swift、文件系统）存储这些镜像文件	glance
	Compute Management	与计算有关的功能（直接与虚拟机软件打交道的功能除外）	nova-api, nova-api-metadata, nova-conductor, nova-consoleauth, nova-novncproxy, nova-scheduler, nova-placement-api, python-novaclient
	Networking Management		neutron-server, neutron-plugin-ml2, python-neutronclient
网络节点	Networking L3 Agent	使用 Linux 的网络协议栈和 iptables 来实现路由、NAT 和浮动 IP 等功能。要求与 L2 Agent 安装在同一台计算机中	neutron-l3-agent
	Networking DHCP Agent	使用第三方软件来实现 DHCP 和 IPv6 的路由通告。要求与 L2 Agent 安装在同一台计算机中	neutron-dhcp-agent

续表

节点名称	服　务	功　能	软　件
网络节点	Networking Metadata Agent	转发虚拟机与控制节点上的 nova-api-metadata 服务之间的数据包。要求与 L2 Agent 安装在同一台计算机中	neutron-metadata-agent
	Networking L2 Agent	通过使用第三方软件来实现二层连通性（类似物理交换机）。一般在网络和计算节点上同时运行	neutron-plugin-linuxbridge-agent | neutron-openvswitch-agent
	Networking ML2 Plug-in	为二层插件模块，它允许 OpenStack 使用那些复杂的二层组网技术，而这些复杂的二层网络技术被广泛应用于真实的数据中心	neutron-plugin-ml2
计算节点	Hypervisor	虚拟机软件	nova-compute-xen | nova-compute-qemu | nova-compute-kvm
	compute	向下与 Hypervisor 打交道，向上与控制节点打交道，上传下达计算命令	nova-compute

Metadata 就是在虚拟机启动时被注入虚拟机中的元数据，包括虚拟机的初始化数据和用户数据，比如机器名、IP 地址等，这些元数据被存放在某个地方，表 8-1 中的 neutron-metadata-agent 和 nova-api-metadata 进程就是用来管理这些元数据的。

接下来分别采用单节点安装、三节点安装和基于容器的自动化安装三种方法搭建云计算实验环境。

8.1 单节点安装

把全部的 OpenStack 组件安装到一台机器中。

8.1.1 系统设计

局域网 192.168.110.0/24 上的一台物理机配置至少需要 8GB 内存、4 核 CPU、1TB 硬盘（固态盘更好），物理机中安装 64 位的 Windows 10 操作系统，再安装 VMware Workstation Pro 12 虚拟机软件。在 VMware 虚拟机软件中创建一台虚拟机，虚拟机中安装 64 位的 ubuntu 16.04 服务器版操作系统。虚拟机的网卡采用桥接模式。单节点架构如图 8-1 所示。

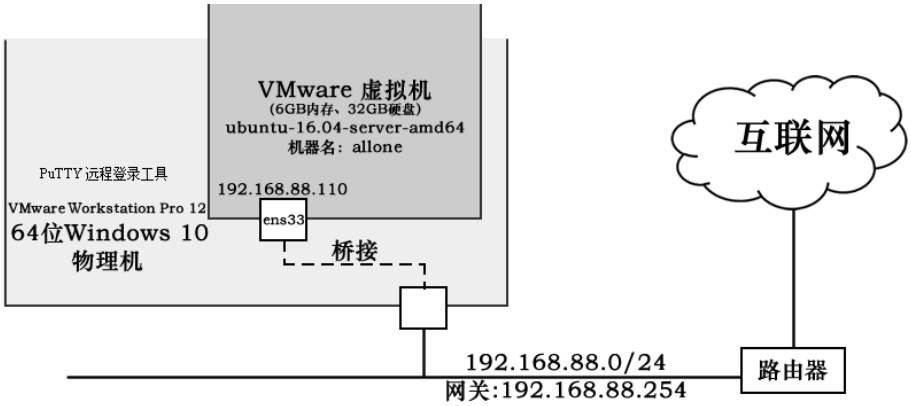


图 8-1 单节点架构

8.1.2 安装过程

1．准备基础环境

创建虚拟机（6GB 内存，2 核 CPU，硬盘 32GB，网卡桥接模式，开启 CPU 的虚拟化特征），在虚拟机里安装 ubuntu-16.04-server-amd64 操作系统，配置好网卡，确保虚拟机能连通互联网，升级系统软件到最新版本，然后依次安装和配置时钟同步软件、数据库软件、消息软件。虚拟机设置如图 8-2 所示。

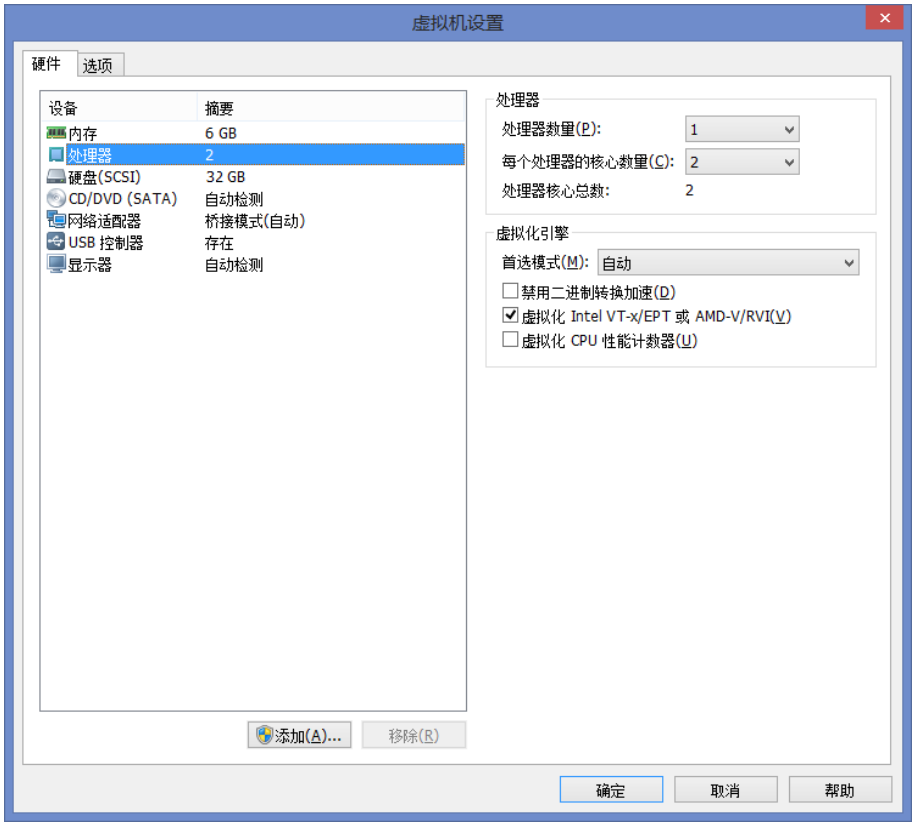


图 8-2　虚拟机设置

以下的操作都是采用Windows系统上的PuTTY工具登录虚拟机的。在虚拟机中安装 Keystone、Glance、Nova、Neutron 和 Horizon 组件。

（1）修改主机名称为 allone。

```
root@OpenStack:~# hostnamectl set-hostname allone
root@OpenStack:~# echo “192.168.88.110   allone” >>/etc/hosts
root@OpenStack:~# shutdown -r now
```

（2）配置网络。

```
root@allone:~# vim /etc/network/interfaces
  # The primary network interface
  auto ens33
  iface ens33 inet static
  address 192.168.88.110
  gateway 192.168.88.254
```

```
    netmask 255.255.255.0
    dns-nameservers 202.96.128.166 8.8.8.8

  root@allone:~# systemctl restart networking

  root@allone:~# ping -c 3 www.bing.com
    PING cn-0001.cn-msedge.net (202.89.233.100) 56(84) bytes of
data.
    64 bytes from 202.89.233.100: icmp_seq=1 ttl=110 time=43.0 ms
    64 bytes from 202.89.233.100: icmp_seq=2 ttl=110 time=43.0 ms
```

上面的最后一条命令测试虚拟机能否连通互联网，本例中表明互联网是连通的。另外网卡的参数要根据实际情况配置。

（3）升级系统软件到最新版。

```
  root@allone:~# apt update -y && apt upgrade -y
  root@allone:~# shutdown -r now
```

默认安装源地址是 http://cn.archive.ubuntu.com/，如果所在的地方网速慢，建议修改文件/etc/apt/sources.list，把安装源地址更换为 http://mirrors.163.com/。

（4）安装时钟同步软件。

```
  root@allone:~# apt install chrony -y
  root@allone:~# systemctl restart chrony
```

配置文件/etc/chrony/chrony.conf 中的默认时钟服务器，2.debian.pool.ntp.org 在国内是可以直接使用的，所以不用修改配置。

（5）安装数据库。

```
  root@allone:~# apt install mariadb-server python-pymysql
  root@allone:~# vim /etc/mysql/mariadb.conf.d/99-OpenStack.cnf
    [mysqld]
    bind-address = 192.168.88.110
    default-storage-engine = innodb
    innodb_file_per_table = on
    max_connections = 4096
    collation-server = utf8_general_ci
    character-set-server = utf8
  root@allone:~# systemctl restart mysql
  root@allone:~# mysql_secure_installation
    ……
```

```
    Enter current password for root (enter for none):
    ……
    Set root password? [Y/n] Y
    New password:MYSQL_PASS
    Re-enter new password:MYSQL_PASS
    Password updated successfully!
    ……

    Remove anonymous users? [Y/n] Y
     ... Success!


    ……
    Disallow root login remotely? [Y/n] n
     ... skipping.
    ……
    Remove test database and access to it? [Y/n] Y
    ……

    Reload privilege tables now? [Y/n] Y
     ... Success!

    All done!  If you've completed all of the above steps, your MariaDB
    installation should now be secure.

    Thanks for using MariaDB!
```

最后一条命令给 MariaDB 数据库的 root 用户设置密码 MYSQL_PASS，另外为了节约篇幅，省略了大量的说明信息。每一个问题后面的“[Y/n]”表示直接按 Enter 键就是回答 Yes，类似的“[y/N]”表示直接按 Enter 键回答 No。

（6）安装消息队列工具。

```
   root@allone:~# apt install rabbitmq-server -y
   #创建消息队列用户OpenStack，并设置密码RABBIT_PASS:
   root@allone:~# rabbitmqctl add_user OpenStack RABBIT_PASS
   #赋予用户OpenStack 配置读、写任何资源的权限:
   root@allone:~# rabbitmqctl set_permissions OpenStack ".*" ".*"
".*"
   root@allone:~# rabbitmqctl list_permissions
     Listing permissions in vhost "/"
     guest   .*      .*      .*
     OpenStack       .*      .*      .*
```

（7）安装 Memcached 工具。

```
root@allone:~# apt install memcached python-memcache -y
root@allone:~# vim /etc/memcached.conf
  ……
  -l 192.168.88.110
  ……
root@allone:~# systemctl restart memcached
```

注意：“-l”是字母 l，其大写是 L，而不是数字 1。

（8）设置 OpenStack 的安装源，并安装 OpenStack 客户端工具。

```
root@allone:~# apt install software-properties-common
root@allone:~# add-apt-repository cloud-archive:queens
  Ubuntu Cloud Archive for OpenStack Queens
  More info: https://wiki.ubuntu.com/ServerTeam/CloudArchive
  Press [ENTER] to continue or ctrl-c to cancel adding it  <-----
这里直接按 Enter 键。

  Reading package lists...
root@allone:~# apt update && apt dist-upgrade -y
root@allone:~# apt install python-openstackclient -y
```

2．安装 OpenStack 的认证服务：Keystone

（1）建立数据库 Keystone 和数据库用户。

```
root@allone:~# mysql
MariaDB [(none)]> CREATE DATABASE keystone;
MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  keystone.*  TO
'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  keystone.*  TO
'keystone'@'%' IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB [(none)]> exit;
```

（2）安装和配置相应的组件。

```
root@allone:~# apt install keystone apache2 libapache2-mod-wsgi -y
root@allone:~# vim /etc/keystone/keystone.conf
  # …
  [database]
  connection                                                    =
mysql+pymysql://keystone:KEYSTONE_DBPASS@allone/keystone
  ……
  [token]
  # ...
  provider = fernet
```

向 Keystone 数据库中导入数据。

```
root@allone:~# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

初始化 fernet 密钥存储库。

```
root@allone:~# keystone-manage fernet_setup --keystone-user
keystone --keystone-group keystone
root@allone:~# keystone-manage credential_setup --keystone-user
keystone --keystone-group keystone
```

初始化身份认证服务。

```
root@allone:~# keystone-manage bootstrap --bootstrap-password
ADMIN_PASS \
  --bootstrap-admin-url http://allone:5000/v3/ \
  --bootstrap-internal-url http://allone:5000/v3/ \
  --bootstrap-public-url http://allone:5000/v3/ \
  --bootstrap-region-id RegionOne
```

配置 Apache。

```
root@allone:~# vim /etc/apache2/apache2.conf
  # …
  ServerName allone
  ……
root@allone:~# systemctl restart apache2
```

配置 Keystone 管理员的环境变量。

```
root@allone:~# export OS_USERNAME=admin
root@allone:~# export OS_PASSWORD=ADMIN_PASS
root@allone:~# export OS_PROJECT_NAME=admin
root@allone:~# export OS_USER_DOMAIN_NAME=Default
root@allone:~# export OS_PROJECT_DOMAIN_NAME=Default
root@allone:~# export OS_AUTH_URL=http://allone:5000/v3
root@allone:~# export OS_IDENTITY_API_VERSION=3
```

（3）创建域、项目、Keystone 用户和角色。

在 Keystone 中默认存在 default 域，允许创建更多的域，比如下面的命令用于创建 example 域。

```
root@allone:~# openstack domain create --description "An Example
Domain" example
  +-------------+----------------------------------+
  | Field       | Value                            |
  +-------------+----------------------------------+
  | description | An Example Domain                |
  | enabled     | True                             |
```

```
     | id          | bf34998091334fbeace02cdf4eb06837 |
     | name        | example                          |
     | tags        | []                               |
     +-------------+----------------------------------+
```

后面的大部分命令都有类似的多行两列的输出，为了节约篇幅，就不再把输出放在正文中。

在 default 域中创建 service 项目用于存放全部的服务。

```
   root@allone:~#  openstack  project  create  --domain  default
--description "Service Project" service
```

创建一个普通项目 demo 和 Keystone 用户 demo，用于完成普通的任务。

```
   root@allone:~# openstack project create --domain default
--description "Demo Project" demo
   root@allone:~#  openstack  user  create  --domain  default
--password-prompt demo
     User Password:DEMO_PASS
     Repeat User Password:DEMO_PASS
   root@allone:~# openstack role create user
   root@allone:~# openstack role add --project demo --user demo user
```

最后一条命令完成授权：即在 demo 项目中授予 demo 用户 user 角色。

（4）验证 Keystone。

先删除几个环境变量。

```
   root@allone:~# unset OS_AUTH_URL OS_PASSWORD
```

获取 Keystone 管理员的令牌。

```
   root@allone:~# openstack --os-auth-url http://allone:5000/v3 \
     --os-project-domain-name Default --os-user-domain-name Default \
     --os-project-name admin --os-username admin token issue
     Password:ADMIN_PASS
```

输入用户 admin 的密码 ADMIN_PASS，如果一切正常就会获得一个令牌。

获取 Keystone 普通用户 demo 的令牌。

```
   root@allone:~# openstack --os-auth-url http://allone:5000/v3 \
     --os-project-domain-name Default --os-user-domain-name Default \
     --os-project-name demo --os-username demo token issue
```

```
  Password:DEMO_PASS
```

输入用户 demo 的密码 DEMO_PASS，如果一切正常就会获得一个令牌。

（5）创建生成环境变量的脚本文件。

产生 admin 用户的环境变量的脚本文件 admin-openrc：

```
root@allone:~# vim admin-openrc
  export OS_PROJECT_DOMAIN_NAME=Default
  export OS_USER_DOMAIN_NAME=Default
  export OS_PROJECT_NAME=admin
  export OS_USERNAME=admin
  export OS_PASSWORD=ADMIN_PASS
  export OS_AUTH_URL=http://allone:5000/v3
  export OS_IDENTITY_API_VERSION=3
  export OS_IMAGE_API_VERSION=2
```

产生 demo 用户的环境变量的脚本文件 demo-openrc：

```
root@allone:~# vim demo-openrc
  export OS_PROJECT_DOMAIN_NAME=Default
  export OS_USER_DOMAIN_NAME=Default
  export OS_PROJECT_NAME=demo
  export OS_USERNAME=demo
  export OS_PASSWORD=DEMO_PASS
  export OS_AUTH_URL=http://allone:5000/v3
  export OS_IDENTITY_API_VERSION=3
  export OS_IMAGE_API_VERSION=2
```

以后就可以“点”执行这些文件来快速产生环境变量，如：

```
root@allone:~# . admin-openrc
root@allone:~# openstack token issue
```

命令“.admin-openrc”就是“点”执行一个脚本文件，这样脚本文件中定义的环境变量就会输出。也可以采用命令“source admin-openrc”，效果是一样的。

3．安装镜像服务：Glance

（1）创建数据库、数据库用户并授权。

```
root@allone:~# mysql -uroot -pMYSQL_PASS
MariaDB [(none)]> CREATE DATABASE glance;
MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  glance.*  TO
```

```
'glance'@'localhost' IDENTIFIED BY 'GLANCE_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%'
IDENTIFIED BY 'GLANCE_DBPASS';
```

（2）创建 Keystone 用户并授权。

```
   # “点”执行脚本程序admin-openrc，从而输出里面定义的环境变量。
   root@allone:~# . admin-openrc
   # 在keystone 中创建用户glance，密码为GLANCE_PASS。
   root@allone:~#  openstack  user  create  --domain  default
--password-prompt glance
     User Password:GLANCE_PASS
     Repeat User Password:GLANCE_PASS
   # 在service 项目中给用户glance 授予管理员权限admin。
   root@allone:~# openstack role add --project service --user
glance admin
   # 创建服务类型为image 的实体 glance。
   root@allone:~#  openstack  service  create  --name  glance
--description "OpenStack Image" image
```

OpenStack 中的服务类型有 compute（计算服务）、image（镜像服务）、identity（身份认证服务）、volume（卷服务）、network（网络服务）和 placement（计算资源调度）等。

（3）创建镜像服务的调用端点。

```
   # 为image 服务类型创建公开（public）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne image
public http://allone:9292
   # 为image 服务类型创建内部（internal）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne image
internal http://allone:9292
   # 为image 服务类型创建管理（admin）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne image
admin http://allone:9292
```

根据权限的不同，OpenStack 把调用端点（endpoint）分为三类，分别是 public、internal 和 admin。

（4）安装并配置相关软件。

```
   root@allone:~# apt install glance -y
   root@allone:~# vim /etc/glance/glance-api.conf

     [database]
     # 配置glance 服务如何访问数据库。
```

```
     connection                                                    =
mysql+pymysql://glance:GLANCE_DBPASS@allone/glance

     [glance_store]
     # 配置镜像文件格式和存放位置：普通文件(file)，存放在
/var/lib/glance/images。
     stores = file,http
     default_store = file
     filesystem_store_datadir = /var/lib/glance/images/

     [keystone_authtoken]
     # glance用户如何访问keystone。
     auth_uri = http://allone:5000
     auth_url = http://allone:5000
     memcached_servers = allone:11211
     auth_type = password
     project_domain_name = Default
     user_domain_name = Default
     project_name = service
     username = glance
     password = GLANCE_PASS

     [paste_deploy]
     flavor = keystone
```

修改文件/etc/glance/glance-api.conf 时要特别注意：上面列出的节[database]、[glance_store]、[keystone_authtoken]和[paste_deploy]中其他的行要添加注释（在行首加“#”字符即可），不要修改文件中的其他节。

```
   root@allone:~# vim /etc/glance/glance-registry.conf

     [database]
     # 配置glance服务如何访问数据库。
     connection                                                    =
mysql+pymysql://glance:GLANCE_DBPASS@allone/glance

     [keystone_authtoken]
     # glance用户如何访问keystone。
     auth_uri = http://allone:5000
     auth_url = http://allone:5000
     memcached_servers = allone:11211
     auth_type = password
     project_domain_name = Default
     user_domain_name = Default
     project_name = service
     username = glance
```

```
password = GLANCE_PASS

[paste_deploy]
flavor = keystone
```

上面列出的节[database]、[keystone_authtoken]和[paste_deploy]中其他的行要添加注释，不要修改文件中的其他节。

（5）向数据库中导入 Glance 的数据。

```
root@allone:~# su -s /bin/sh -c "glance-manage db_sync" glance
```

（6）启动 Glance 服务。

```
root@allone:~# systemctl restart glance-api
root@allone:~# systemctl restart glance-registry
```

（7）验证。

```
root@allone:~# . admin-openrc

# 把文件cirros-0.4.0-x86_64-disk.img 下载到当前目录中。
root@allone:~# wget
http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.im
g

# 使用刚刚下载的文件创建镜像cirros。
root@allone:~# openstack image create "cirros" --file
cirros-0.4.0-x86_64-disk.img --disk-format qcow2 --container-format
bare --public

root@allone:~# openstack image list
  +--------------------------------------+--------+--------+
  | ID                                   | Name   | Status |
  +--------------------------------------+--------+--------+
  | 18ecc4f2-078a-4d05-bb5d-d2c123e2429d | cirros | active |
  +--------------------------------------+--------+--------+
```

如果看到上面最后一条命令的输出，就说明 Glance 服务已成功安装。

4．安装计算服务：Nova

（1）创建数据库、数据库用户并授权。

```
root@allone:~# mysql -uroot -pMYSQL_PASS
MariaDB [(none)]> CREATE DATABASE nova_api;
```

```
   MariaDB [(none)]> CREATE DATABASE nova;
   MariaDB [(none)]> CREATE DATABASE nova_cell0;
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
```

（2）建立 Nova 用户并授权。

```
   # “点”执行脚本程序 admin-openrc，从而输出里面定义的环境变量。如果没有注
销 Linux 的用户，环境变量一直存在，也可以不“点”执行。
   root@allone:~# . admin-openrc
   # 在 keystone 中创建用户 nova，密码为 NOVA_PASS。
   root@allone:~#  openstack  user  create  --domain  default
--password-prompt nova
     User Password:NOVA_PASS
     Repeat User Password:NOVA_PASS
   # 在 service 项目中给用户 nova 授予管理员权限 admin。
   root@allone:~# openstack role add --project service --user nova
admin
   # 创建服务类型为 compute 的实体 nova。
   root@allone:~# openstack service create --name nova --description
"OpenStack Compute" compute
```

（3）创建计算服务的调用端点。

```
   # 为compute 服务类型创建公开（public）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
compute public http://allone:8774/v2.1
   # 为compute 服务类型创建内部（internal）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
compute internal http://allone:8774/v2.1
   # 为compute 服务类型创建管理（admin）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
compute admin http://allone:8774/v2.1
```

（4）建立 placement 用户并授权。

```
   # 在 keystone 中创建用户 placement，密码为 PLACEMENT_PASS。
   root@allone:~#  openstack  user  create  --domain  default
--password-prompt placement
```

```
     User Password:PALCEMENT_PASS
     Repeat User Password:PLACEMENT_PASS
   # 在 service 项目中给用户 placement 授予管理员权限 admin。
   root@allone:~#  openstack  role  add  --project  service  --user
placement admin
   # 创建服务类型为 placement 的实体 placement。
   root@allone:~#  openstack  service  create  --name  placement
--description "Placement API" placement
```

（5）创建相应的调用端点。

```
   # 为placement 服务类型创建公开（public）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
placement public http://allone:8778
   # 为placement 服务类型创建内部（internal）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
placement internal http://allone:8778
   # 为placement 服务类型创建管理（admin）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
placement admin http://allone:8778
```

（6）安装并配置相关软件。

```
   root@allone:~#    apt    install    nova-api    nova-conductor
nova-consoleauth nova-novncproxy nova-scheduler nova-placement-api
nova-compute -y
   root@allone:~# vim /etc/nova/nova.conf
     [DEFAULT]
     lock_path = /var/lock/nova
     state_path = /var/lib/nova
     transport_url = rabbit://OpenStack:RABBIT_PASS@allone
     my_ip = 192.168.88.110
     use_neutron = True
     firewall_driver = nova.virt.firewall.NoopFirewallDriver

     [api]
     auth_strategy = keystone

     [api_database]
     connection = mysql+pymysql://nova:NOVA_DBPASS@allone/nova_api

     [cells]
     enable = False

     [database]
     connection = mysql+pymysql://nova:NOVA_DBPASS@allone/nova

     [glance]
     api_servers = http://allone:9292
```

```
[keystone_authtoken]
auth_url = http://allone:5000/v3
memcached_servers = allone:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = NOVA_PASS

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://allone:5000/v3
username = placement
password = PLACEMENT_PASS

[vnc]
enabled = true
server_listen = $my_ip
server_proxyclient_address = $my_ip
novncproxy_base_url = http://allone:6080/vnc_auto.html
```

修改文件/etc/nova/nova.conf 时需要注意：上面列出的节中，其他的行要全部添加注释或删除，文件中的其他节不要改动。

（7）向数据库中导入 nova、nova_api、nova_cell0 相关的数据。

```
root@allone:~# su -s /bin/sh -c "nova-manage api_db sync" nova
root@allone:~# su -s /bin/sh -c "nova-manage cell_v2 map_cell0" nova
root@allone:~# su -s /bin/sh -c "nova-manage cell_v2 create_cell
--name=cell1 --verbose" nova
root@allone:~# su -s /bin/sh -c "nova-manage db sync" nova
```

（8）检查 cell0 和 cell1 是否注册成功。

```
root@allone:~# nova-manage cell_v2 list_cells
  +-------+--------------------------------------+
  |  Name  |                 UUID                 |
  +-------+--------------------------------------+
  | cell0 | 00000000-0000-0000-0000-000000000000 |
```

```
     | cell1 | e9ff2279-a352-44e4-bef6-c4292f187091 |
     +-------+--------------------------------------+
```

（9）检查机器的 CPU 是否支持虚拟化。

```
   root@allone:~# egrep -c '(vmx|svm)' /proc/cpuinfo
     2
```

如果命令返回的数字不大于 0，说明 CPU 不支持虚拟化，这时就要修改文件/etc/nova/nova-compute.conf，把里面的参数改为“virt_type=qemu”，然后采用命令 systemctl restart nova-compute 重启 nova-compute 服务。不过在我们的实验环境中，VMware 虚拟机即使开启了 CPU 的虚拟化特征，在以后创建虚拟机时还是不能正常运行，所以一定要修改为“virt_type=qemu”。

（10）把计算节点添加到 cell 数据库。

```
   root@allone:~# . admin-openrc
   root@allone:~#   openstack   compute   service   list   --service
nova-compute
     +----+--------------+--------+------+---------+-------+------
----------------------+
     | ID | Binary       | Host   | Zone | Status  | State | Updated At
|
     +----+--------------+--------+------+---------+-------+------
----------------------+
     |   6  |  nova-compute  |  allone  |  nova  |  enabled  |  up     |
2018-07-25T04:32:57.000000  |
     +----+--------------+--------+------+---------+-------+------
----------------------+
   root@allone:~# su -s /bin/sh -c "nova-manage cell_v2 discover_hosts
--verbose" nova
     Found 2 cell mappings.
     Skipping cell0 since it does not contain hosts.
     Getting       computes       from       cell       'cell1':
e9ff2279-a352-44e4-bef6-c4292f187091
     Checking   host   mapping   for   compute   host   'allone':
44fd740b-6a53-4f6d-9bb5-d140ca28d059
     Creating   host   mapping   for   compute   host   'allone':
44fd740b-6a53-4f6d-9bb5-d140ca28d059
     Found      1      unmapped      computes      in      cell:
e9ff2279-a352-44e4-bef6-c4292f187091
```

每次新增加了计算节点，就要在控制节点上运行 su -s /bin/sh -c "nova-manage cell_v2 discover_hosts --verbose" nova 命令，本实验中，控制节点就是

计算节点。也可以修改文件/etc/nova/nova.conf，在“[scheduler]”节下添加参数：discover_hosts_in_cells_interval = 300，这样每隔 300 秒就会自动发现新增加的计算节点。

（11）重启所有 Nova 服务。

```
root@allone:~# shutdown -r now
root@allone:~# systemctl stop nova-api
root@allone:~# systemctl stop nova-consoleauth
root@allone:~# systemctl stop nova-scheduler
root@allone:~# systemctl stop nova-conductor
root@allone:~# systemctl stop nova-novncproxy
root@allone:~# systemctl stop nova-compute

root@allone:~# rm /var/log/nova/*

root@allone:~# systemctl start nova-api
root@allone:~# systemctl start nova-consoleauth
root@allone:~# systemctl start nova-scheduler
root@allone:~# systemctl start nova-conductor
root@allone:~# systemctl start nova-novncproxy
root@allone:~# systemctl start nova-compute

root@allone:~# grep ERROR /var/log/nova/*
```

如果上面最后一条命令有输出，说明 Nova 的某些服务存在问题，需要根据具体的日志排查问题。最常见的错误是 nova-compute 服务与消息队列连接失败。排查的命令有 rabbitmqctl list_permissions（查看消息队列的用户和权限）、OpenStack user list（列出 Keystone 中的用户）、OpenStack service list（列出 Keystone 中的服务）、OpenStack endpoint list（列出全部的调用端点）。正常的输出如下。

```
root@allone:~# rabbitmqctl list_permissions
  Listing permissions in vhost "/"
  guest   .*      .*      .*
  OpenStack       .*      .*      .*

root@allone:~# . admin-openrc
root@allone:~# openstack user list
  +----------------------------------+-----------+
  | ID                               | Name      |
  +----------------------------------+-----------+
  | 191b931af7b14291a46921025ec4a6d6 | glance    |
```

```
     | cac24f95fa2443aabb06afd24b15e6b3 | placement |
     | e213b38ad7b040d6beb24ced770d86ce | admin     |
     | e9c18b26d95f4913916ba926aabdd772 | demo      |
     | ebb9c4e4c17b4cdf88c1e69d054086ca | nova      |
     +----------------------------------+-----------+

   root@allone:~# openstack service list
     +----------------------------------+-----------+-----------+
     | ID                               | Name      | Type      |
     +----------------------------------+-----------+-----------+
     | 064efe9307a54c5db965205e173a0cf0 | nova      | compute   |
     | 2565fa5c52c8492f9126a143e1d257ea | keystone  | identity  |
     | 6331f51c824e42a389afeeceddc76144 | placement | placement |
     | b62e16e8f5194843aa2261bbe724e0b4 | glance    | image     |
     +----------------------------------+-----------+-----------+

   root@allone:~# openstack endpoint list
      +----+-----------+----------+------+-----------+------------
-------------+
      |  ID  |  Service  Na|  Service  T|  Enabl|  Interface  |  URL
|
      +----+-----------+----------+------+-----------+------------
-------------+
      |  0c2|  placement  |  placement|  True  |  internal    |
http://allone:8778      |
      |  424|  nova         |  compute   |  True  |  internal   |
http://allone:8774/v2.1  |
      |  47f|  glance      |  image      |  True  |  internal   |
http://allone:9292       |
      |  54f|  glance      |  image      |  True  |  public     |
http://allone:9292       |
      |  61c|  placement  |  placement|  True  |  public       |
http://allone:8778      |
      |  761|  keystone    |  identity  |  True  |  internal    |
http://allone:5000/v3/  |
      |  77f|  glance      |  image      |  True  |  admin       |
http://allone:9292       |
      |  a11|  keystone    |  identity  |  True  |  admin       |
http://allone:5000/v3/  |
      |  aa5|  nova         |  compute   |  True  |  public      |
http://allone:8774/v2.1  |
      |  bfd|  placement  |  placement|  True  |  admin        |
http://allone:8778     |
      |  e4f|  keystone    |  identity  |  True  |  public      |
http://allone:5000/v3/  |
      |  fc2|  nova         |  compute   |  True  |  admin       |
http://allone:8774/v2.1  |
      +----+-----------+----------+------+-----------+------------
```

```
-------------+

   # 下面这条命令列出计算机中正在被监听的端口号，必须要包含上面那条命令输出的最
后一列中的端口号。
   root@allone:~# netstat -tnulp
      Active Internet connections (only servers)
      Proto  Local Address          Foreign Addr  State   PID/Program
name
      tcp       192.168.88.110:3306         0.0.0.0:*         LISTEN
1798/mysqld
      tcp       192.168.88.110:11211        0.0.0.0:*         LISTEN
1082/memcached
      tcp      0.0.0.0:9292                  0.0.0.0:*         LISTEN
2438/python2
      tcp    0.0.0.0:4369          0.0.0.0:*     LISTEN  1492/epmd
      tcp      192.168.122.1:53             0.0.0.0:*         LISTEN
2384/dnsmasq
      tcp    0.0.0.0:22            0.0.0.0:*     LISTEN  1087/sshd
      tcp      0.0.0.0:6080                  0.0.0.0:*         LISTEN
2434/python2
      tcp      0.0.0.0:8774                  0.0.0.0:*         LISTEN
2430/python2
      tcp      0.0.0.0:8775                  0.0.0.0:*         LISTEN
2430/python2
      tcp      0.0.0.0:9191                  0.0.0.0:*         LISTEN
2435/python2
      tcp      0.0.0.0:25672                 0.0.0.0:*         LISTEN
1989/beam.smp
      tcp6   :::8778               :::*         LISTEN  1835/apache2
      tcp6   :::80                 :::*         LISTEN  1835/apache2
      tcp6   :::4369               :::*         LISTEN  1492/epmd
      tcp6    :::5672                      :::*              LISTEN
1989/beam.smp
      tcp6   :::5000               :::*         LISTEN  1835/apache2
      udp           192.168.122.1:53                       0.0.0.0:*
2384/dnsmasq
      udp    0.0.0.0:67            0.0.0.0:*           2384/dnsmasq
      udp    127.0.0.1:323          0.0.0.0:*           1468/chronyd
      udp6   ::1:323               :::*                1468/chronyd

   # 列出全部的计算服务。
   root@allone:~# openstack compute service list
      +----+------------------+--------+----------+---------+-----
--+------------+
      | ID | Binary          | Host   | Zone     | Status  | State |
Updated At |
      +----+------------------+--------+----------+---------+-----
--+------------+
```

```
      |  3 | nova-consoleauth | allone | internal | enabled | up    |
2018-07-24 |
      |  4 | nova-scheduler   | allone | internal | enabled | up    |
2018-07-24 |
      |  5 | nova-conductor   | allone | internal | enabled | up    |
2018-07-24 |
      |  6 | nova-compute     | allone | nova     | enabled | up    |
2018-07-24 |
      +----+------------------+--------+----------+---------+-----
--+------------+

   root@allone:~# openstack catalog list
      +-----------+-----------+-----------------------------------+
      | Name      | Type      | Endpoints                         |
      +-----------+-----------+-----------------------------------+
      | nova      | compute   | RegionOne                         |
      |           |           |   internal: http://allone:8774/v2.1 |
      |           |           | RegionOne                         |
      |           |           |   public: http://allone:8774/v2.1   |
      |           |           | RegionOne                         |
      |           |           |   admin: http://allone:8774/v2.1    |
      |           |           |                                   |
      | keystone  | identity  | RegionOne                         |
      |           |           |   internal: http://allone:5000/v3/  |
      |           |           | RegionOne                         |
      |           |           |   admin: http://allone:5000/v3/     |
      |           |           | RegionOne                         |
      |           |           |   public: http://allone:5000/v3/    |
      |           |           |                                   |
      | placement | placement | RegionOne                         |
      |           |           |   internal: http://allone:8778      |
      |           |           | RegionOne                         |
      |           |           |   public: http://allone:8778        |
      |           |           | RegionOne                         |
      |           |           |   admin: http://allone:8778         |
      |           |           |                                   |
      | neutron   | network   | RegionOne                         |
      |           |           |   internal: http://allone:9696      |
      |           |           | RegionOne                         |
      |           |           |   admin: http://allone:9696         |
      |           |           | RegionOne                         |
      |           |           |   public: http://allone:9696        |
      |           |           |                                   |
      | glance    | image     | RegionOne                         |
      |           |           |   internal: http://allone:9292      |
      |           |           | RegionOne                         |
      |           |           |   public: http://allone:9292        |
      |           |           | RegionOne                         |
```

```
      |          |          |  admin: http://allone:9292          |
      |          |          |                                     |
      +----------+----------+-------------------------------------+

   # 检查cell和placement API工作是否正常。
   root@allone:~# nova-status upgrade check
     +--------------------------------+
     | Upgrade Check Results         |
     +--------------------------------+
     | Check: Cells v2                |
     | Result: Success                |
     | Details: None                 |
     +--------------------------------+
     | Check: Placement API           |
     | Result: Success                |
     | Details: None                 |
     +--------------------------------+
     | Check: Resource Providers      |
     | Result: Success                |
     | Details: None                 |
     +--------------------------------+
```

5．安装网络服务：Neutron

（1）创建数据库、数据库用户并授权。

```
   root@allone:~# mysql -uroot -pMYSQL_PASS
   MariaDB [(none)]> CREATE DATABASE neutron;
   MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  neutron.*  TO
'neutron'@'localhost' IDENTIFIED BY 'NEUTRON_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%'
IDENTIFIED BY 'NEUTRON_DBPASS';
```

（2）建立 Keystone 用户并授权。

```
   # “点”执行脚本程序admin-openrc，从而输出里面定义的环境变量。
   root@allone:~# . admin-openrc
   # 在keystone中创建用户neutron，密码为NEUTRON_PASS。
   root@allone:~#   openstack   user   create   --domain   default
--password-prompt neutron
     User Password:NEUTRON_PASS
     Repeat User Password:NEUTRON_PASS
   # 在service项目中给用户neutron授予管理员权限admin。
   root@allone:~# openstack role add --project service --user neutron
admin
   # 创建服务类型为network的实体 neutron。
   root@allone:~#   openstack   service   create   --name   neutron
--description "OpenStack Networking" network
```

（3）创建镜像服务的调用端点。

```
   # 为network服务类型创建公开（public）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
network public http://allone:9696
   # 为network服务类型创建内部（internal）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
network internal http://allone:9696
   # 为network服务类型创建管理（admin）的调用端点。
   root@allone:~# openstack endpoint create --region RegionOne
network admin http://allone:9696
```

本实验采用 provider network（供应商网络）模式和 Linux 自带的网桥，所以虚拟机的 IP 地址就取自物理机所在的局域网。其他的选项还有 self-service network（租户网络）模式、Open vSwitch 网桥等。

（4）安装并配置相关软件。

```
   root@allone:~# apt install neutron-server neutron-plugin-ml2
neutron-linuxbridge-agent neutron-dhcp-agent neutron-metadata-agent
-y

   root@allone:~# vim /etc/neutron/neutron.conf

     [DEFAULT]
     core_plugin = ml2
     service_plugins =
     transport_url = rabbit://OpenStack:RABBIT_PASS@allone
     auth_strategy = keystone
     notify_nova_on_port_status_changes = true
     notify_nova_on_port_data_changes = true

     [database]
     # 配置neutron服务如何访问数据库。
     connection                                                    =
mysql+pymysql://neutron:NEUTRON_DBPASS@allone/neutron

     [keystone_authtoken]
     # neutron用户如何访问keystone。
     auth_uri = http://allone:5000
     auth_url = http://allone:5000
     memcached_servers = allone:11211
     auth_type = password
     project_domain_name = default
     user_domain_name = default
     project_name = service
     username = neutron
```

```
password = NEUTRON_PASS

[nova]
auth_url = http://allone:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

修改文件/etc/neutron/neutron.conf 时要特别注意：上面列出的节[DEFAULT]、[database]、[keystone_authtoken]和[nova]中其他的行要添加注释，不要修改文件中的其他节。

```
root@allone:~# vim /etc/neutron/plugins/ml2/ml2_conf.ini

[ml2]
type_drivers = local,flat,vlan
tenant_network_type =
mechanism_drivers = linuxbridge
extension_drivers = port_security

[ml2_type_flat]
flat_networks = my_physical_net

[securitygroup]
enable_ipset = true
```

上面列出的节[ml2]、[ml2_type_flat]和[securitygroup]中其他的行要添加注释，不要修改文件中的其他节。

配置 Linux 桥接。

```
root@allone:~# vim /etc/neutron/plugins/ml2/ linuxbridge_agent.ini

[linux_bridge]
physical_interface_mappings = my_physical_net:ens33
[securitygroup]
enable_security_group = true
firewall_driver    =    neutron.agent.linux.iptables_firewall.
IptablesFirewallDriver
[vxlan]
enable_vxlan = false
```

注意：上面的 ens33 是 Linux 系统中网卡的名字，可以使用命令 ifconfig 查看。

在 provider network 模式下，DHCP 主要用于完成虚拟机访问元数据。

```
root@allone:~# vim /etc/neutron/dhcp_agent.ini

[DEFAULT]
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

配置管理元数据的服务器和访问密码。

```
root@allone:~# vim /etc/neutron/metadata_agent.ini

  [DEFAULT]
  nova_metadata_host = allone
  metadata_proxy_shared_secret = METADATA_PASS
```

配置计算服务访问网络服务。

```
root@allone:~# vim /etc/nova/nova.conf

  [neutron]
  url = http://allone:9696
  auth_url = http://allone:5000
  auth_type = password
  project_domain_name = default
  user_domain_name = default
  region_name = RegionOne
  project_name = service
  username = neutron
  password = NEUTRON_PASS
  service_metadata_proxy = true
  metadata_proxy_shared_secret = METADATA_PASS
```

文件/etc/nova/nova.conf 中其他的节不要修改。

（5）开启 Linux 内核的桥接功能。

确保 Linux 内核参数 net.bridge.bridge-nf-call-iptables、net.bridge.bridge-nf-call-ip6tables 的值为 1。

```
root@allone:~# sysctl net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-iptables = 1
```

```
root@allone:~# sysctl net.bridge.bridge-nf-call-ip6tables
net.bridge.bridge-nf-call-ip6tables = 1

# 如果不为1，就采用下面的命令修改。
root@allone:~# sysctl -w net.bridge.bridge-nf-call-iptables=1
root@allone:~# sysctl -w net.bridge.bridge-nf-call-ip6tables=1
```

（6）向数据库中导入 Neutron 的数据。

```
root@allone:~# su -s /bin/sh -c "neutron-db-manage --config-file
/etc/neutron/neutron.conf --config-file
/etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron
```

（7）启动有关服务。

```
root@allone:~# systemctl restart nova-api
root@allone:~# systemctl restart nova-compute

root@allone:~# systemctl stop neutron-server
root@allone:~# systemctl stop neutron-linuxbridge-agent
root@allone:~# systemctl stop neutron-dhcp-agent
root@allone:~# systemctl stop neutron-metadata-agent

root@allone:~# rm /var/log/neutron/*

root@allone:~# systemctl start neutron-server
root@allone:~# systemctl start neutron-linuxbridge-agent
root@allone:~# systemctl start neutron-dhcp-agent
root@allone:~# systemctl start neutron-metadata-agent

root@allone:~# openstack network agent list
  +----+--------------------+--------+-----+----+--------------
-------------+
  | ID | Agent Type       | Host  | Aliv| Sta| Binary              |
  +----+--------------------+--------+-----+----+--------------
-------------+
  |  11e|  Metadata  agent        |  allone  |  :-)  |  UP  |
neutron-metadata-agent   |
  | 830| DHCP agent    | allone | :-) | UP | neutron-dhcp-agent   |
  |  a7e|  Linux  bridge  agent  |  allone  |  :-)  |  UP  |
neutron-linuxbridge-agent |
  +----+--------------------+--------+-----+----+--------------
-------------+
```

如果看到类似上面最后一条命令的输出，说明 Neutron 服务安装成功，可以再查看 9696 端口是否被监听，进行进一步确认。如果没有输出，等 1 分钟左右再运行最后一条命令。

8.1.3 启动一台虚拟机

现在一个最小的云端完成了，接下来采用命令方式创建并启动一台虚拟机。等以后安装了 Horizon 组件之后，就可以使用 Web 图形工具来管理云端了。

1. 创建虚拟网络

```
root@allone:~# shutdown -r now
root@allone:~# . admin-openrc

# 创建类型为provider、名字为provider-net1的网络。
root@allone:~# openstack network create --share --external
--provider-physical-network my_physical_net --provider-network-type
flat provider-net1

# 在provider-net1上创建一个子网provider-subnet1，子网内共有20个IP
地址（从192.168.88.64至192.168.88.83），IP地址和网关都是物理机器所在的局域
网。
root@allone:~# openstack subnet create --network provider-net1 \
  --allocation-pool start=192.168.88.64,end=192.168.88.83 \
  --dns-nameserver 8.8.4.4 --gateway 192.168.88.254 \
  --subnet-range 192.168.88.0/24 provider-subnet1
```

注意：子网内 IP 数至少需要 2 个，否则没法启动虚拟机。

2. 创建一个虚拟机类型

```
root@allone:~# openstack flavor create --id 0 --vcpus 1 --ram 64
--disk 1 m1.nano
```

3. 创建一对密钥

```
root@allone:~# . demo-openrc
root@allone:~# ssh-keygen -q -N ""
  Enter file in which to save the key (/root/.ssh/id_rsa):直接按
Enter键

root@allone:~#    openstack    keypair    create    --public-key
~/.ssh/id_rsa.pub mykey

# 显示刚刚创建的密钥对。
root@allone:~# openstack keypair list
  +-------+-------------------------------------------------+
  | Name  | Fingerprint                                     |
  +-------+-------------------------------------------------+
```

```
     | mykey | 36:71:b4:73:11:2e:50:04:96:f1:19:6f:88:4b:21:82 |
     +-------+--------------------------------------------------+
```

4．添加安全组规则，允许 ping 和 SSH 到虚拟机

```
   root@allone:~# openstack security group rule create --proto icmp
default
   root@allone:~# openstack security group rule create --proto tcp
--dst-port 22 default
```

5．检查启动虚拟机的各种条件

```
   root@allone:~# . ~/demo-openrc

   root@allone:~# openstack flavor list
     +----+---------+-----+------+-----------+-------+-----------+
     | ID | Name    | RAM  | Disk | Ephemeral | VCPUs | Is Public |
     +----+---------+-----+------+-----------+-------+-----------+
     | 0  | m1.nano |  64  |    1 |         0 |     1 | True      |
     +----+---------+-----+------+-----------+-------+-----------+

   root@allone:~# openstack image list
     +--------------------------------------+--------+--------+
     | ID                                   | Name   | Status |
     +--------------------------------------+--------+--------+
     | 18ecc4f2-078a-4d05-bb5d-d2c123e2429d | cirros | active |
     +--------------------------------------+--------+--------+

   root@allone:~# openstack network list
       +--------------------------------------+--------------+---
----------------------------------+
       | ID                                   | Name       | Subnets       |
       +--------------------------------------+--------------+---
----------------------------------+
       |  9cc3c0c8-49b9-48d9-b891-b4c76e7d99f5  |  provider-net1  |
ccc639ca-6cca-4157-9cd3-f518a492c473 |
       +--------------------------------------+--------------+---
----------------------------------+

   root@allone:~# openstack security group list
       +--------------------------------------+---------+---------
--------------+--------+
       | ID                                   | Name    | Description
| Project|
       +--------------------------------------+---------+---------
--------------+--------+
       | 6bc021c7-6eb1-40d0-af5c-0868a606e80f | default | Default
security group | 0c255ae|
       +--------------------------------------+---------+---------
```

```
--------------+--------+
```

6．创建并启动虚拟机

```
   root@allone:~# openstack server create --flavor m1.nano --image
cirros \
     --nic              net-id=9cc3c0c8-49b9-48d9-b891-b4c76e7d99f5
--security-group default \
     --key-name mykey vm1
```

7．查看虚拟机状态

```
   root@allone:~# openstack server list
      +--------+------+--------+----------------------------+----
----+---------+
      | ID     | Name | Status | Networks                   | Image  |
Flavor  |
      +--------+------+--------+----------------------------+----
----+---------+
      | 7b0... | vm1  | ACTIVE | provider-net1=192.168.88.73 | cirros
| m1.nano |
      +--------+------+--------+----------------------------+----
----+---------+
```

可以看到新创建的虚拟机的 IP 地址是 192.168.88.73。下面我们来登录虚拟机。

8．登录虚拟机

通过 Web 控制台登录虚拟机。

```
   root@allone:~# openstack console url show vm1
      +-------+-----------------------------------------------+
      | Field | Value                                         |
      +-------+-----------------------------------------------+
      | type  | novnc                                         |
      |  url     |  http://allone:6080/vnc_auto.html?token=5089caa2-
2a33-4ea7-9782-0437409ea638 |
      +-------+-----------------------------------------------+
```

然后在其他计算机上用浏览器打开网址 http://allone:6080/vnc_auto.html?token=5089caa2-2a33-4ea7-9782-0437409ea638 即可进入虚拟机的控制台，输入用户 cirros 和密码 gocubsgo 即可登录虚拟机。如果不能解释域名 allone 就无法打开这个网站，有两种方法可以解决这个问题：一是直接用 IP 地址

192.168.88.110 替换网址中的 allone，或者在 c:\windows\system32\drivers\etc\hosts 中添加下面一行：

```
192.168.88.110   allone
```

通过 SSH 登录虚拟机：

```
root@allone:~# ssh cirros@192.168.88.73
cirros@192.168.88.66's password:
$
```

输入密码 gocubsgo 即可登录。

到此一个最小的云端算是搭建完毕了，只能采用命令来管理，下面继续安装图形化管理工具 Horizon。

8.1.4 安装 Horizon

1. 安装和配置

```
root@allone:~# apt install openstack-dashboard

# 修改一些参数的值。
root@allone:~# vim /etc/openstack-dashboard/local_settings.py

  OPENSTACK_HOST = "allone"

  OPENSTACK_KEYSTONE_URL = "http://%s:5000/v3" % OPENSTACK_HOST

         'LOCATION': 'allone:11211',

  OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True

  OPENSTACK_API_VERSIONS = {
      "identity": 3,
      "image": 2,
      "volume": 2,
      "compute": 2,
  }

  OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = 'Default'

  OPENSTACK_NEUTRON_NETWORK = {
      'enable_router': False,
      'enable_quotas': False,
      'enable_ipv6': False,
      'enable_distributed_router': False,
      'enable_ha_router': False,
      'enable_fip_topology_check': False,
```

```
TIME_ZONE = "Asia/Shanghai"
```

注意：修改/etc/OpenStack-dashboard/local_settings.py 文件时强烈建议先搜索参数名，然后再修改参数的值，比如在 VIM 中搜索参数名称 OPENSTACK_HOST，找到后把它的值改为“alone”即可。另外，再检查文件/etc/apache2/conf-available/OpenStack-dashboard.conf 有无如下一行，如果没有就添加：

```
WSGIApplicationGroup %{GLOBAL}
```

最后重新加载 Apache：

```
root@allone:~# systemctl reload apache2

# 或者直接重启 apache2。
root@allone:~# systemctl restart apache2
```

2．访问 Web 管理页面

在任何一台计算机上用浏览器打开网址 http://allone/horizon，如果无法解析域名 allone，就打开网址 http://192.168.88.110/horizon，这样就看到如图 8-3 所示的登录界面。

图 8-3　Horizon 登录界面

可以使用用户 admin（密码是 ADMIN_PASS）或者 demo（密码是 DEMO_PASS）登录。登录之后看到的管理界面，如图 8-4 所示。

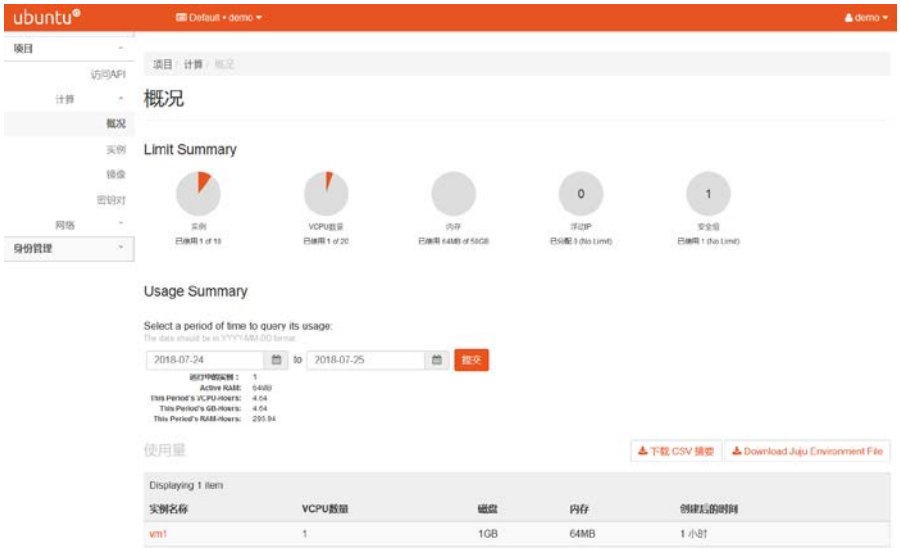


图 8-4　Horizon 管理界面

admin 用户是云端管理员，登录之后的页面具有更多的功能，而 demo 是一个普通的租户，只能做一些权限范围之内的操作。

8.2　三节点安装

控制节点、网络节点和计算节点分别对应一台机器。

8.2.1　系统设计

由于要在 VMware 上完成实验，最终的系统架构如图 8-5 所示。如果是采用实际的物理机，需要进行小小的改动：控制节点不需要接入提供商网络，当然接入也无大碍。之所以在本实验中控制节点要接入提供商网络，目的就是可以在控制节点上 ping 或者远程登录以后创建的 OpenStack 虚拟机(做

实验的三台虚拟机称为 VMware 虚拟机，以作区分）。

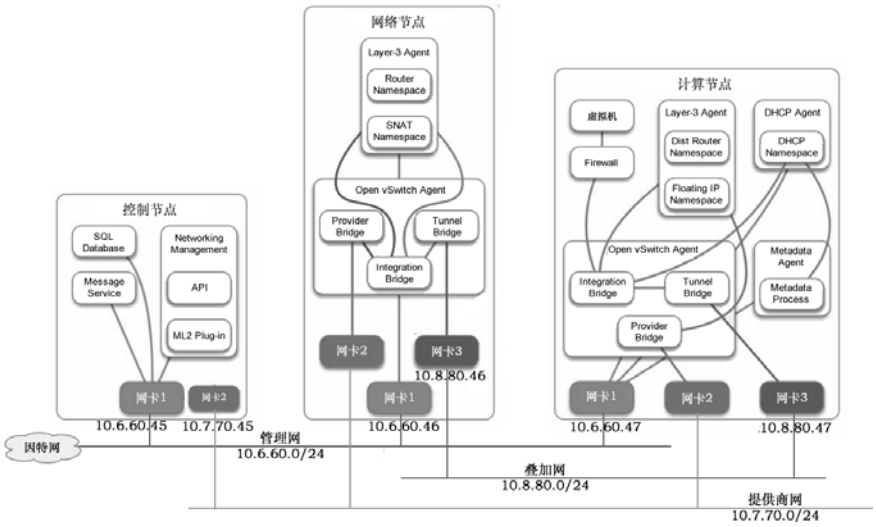


图 8-5　带 DVR 的三节点部署架构

这次换一台硬件配置更好的计算机来做实验，物理机上还是安装 64 位的 Windows 10 操作系统和 VMware Workstation Pro 12 虚拟机软件，然后创建三台 VMware 虚拟机，分别充当 OpenStack 的控制节点、网络节点和计算节点，这些虚拟机的配置如图 8-6 所示。由图 8-7 可知，每台 VMware 虚拟机用一块网卡桥接到物理机的网卡上，这样 VMware 虚拟机网卡的 IP 地址就落在物理机所在的局域网中，即在本实验环境中只能取自 10.6.60.0/24 网络，而且 IP 地址不能相同。物理机所在的局域网 10.6.60.0/24 就是提供商网络，即 provider network。另外再给每个节点各增加一块网卡，网卡模式为“仅主机模式”，统一接到 VMware 的 VMnet1 桥上，配置 VMnet1 的方法是：在 VMware WorkStation Pro 虚拟机软件中，单击菜单“编辑→虚拟网络编辑器”，在弹出的窗口中单击右下部的“更改设置”以获取管理员权限，如图 8-8 所示，子网 IP 设为 10.7.70.0/24 网络。最后，网络节点和计算节点额外再增加一块网卡组成叠加网络，专门供 OpenStack 虚拟机之间通信，网卡的模式也设为“仅主机模式”，统一接到 VMware 的 VMnet2 桥上，网络为 10.8.80.0/24。

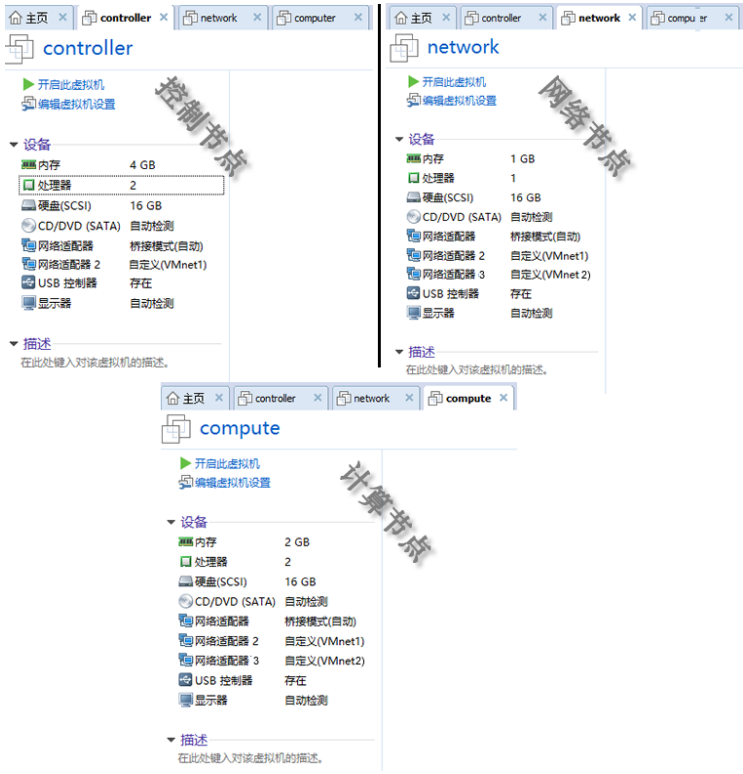


图 8-6　三台虚拟机的配置

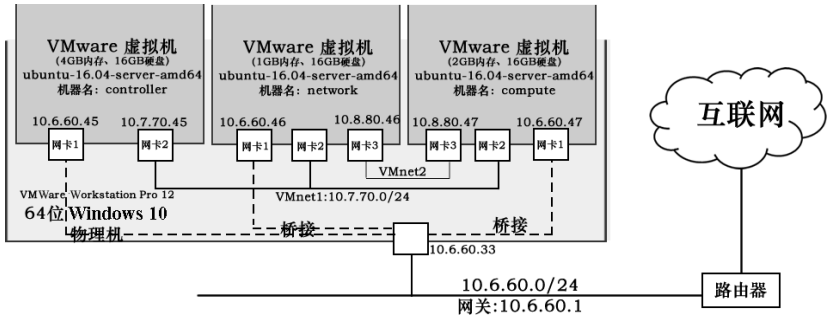


图 8-7　网络拓扑结构

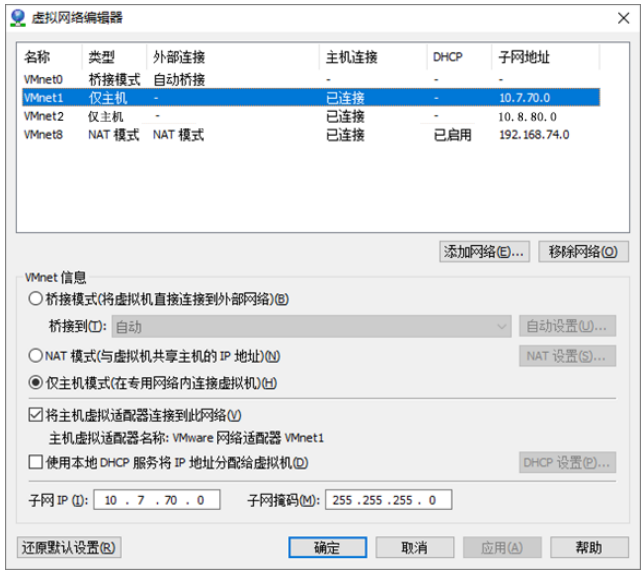


图 8-8　编辑 VMware 虚拟机网络

如图 8-9 所示，查看网卡的 MAC 地址，如果 MAC 地址为空白，则单击“生成(G)”按钮产生新的 MAC 地址。

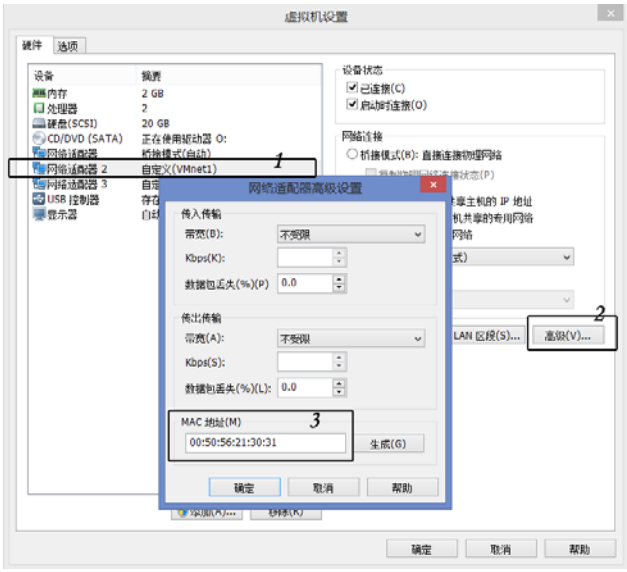


图 8-9　在 VMware 中查看网卡的 MAC 地址

8.2.2 安装过程

后续安装过程由于涉及三台机器，可通过命令提示符作区分，规定：

```
root@controller:~# 命令
```

表示在控制节点上执行“命令”。

```
root@network:~# 命令
```

表示在网络节点上执行“命令”。

```
root@compute:~# 命令
```

表示在计算节点上执行“命令”。

```
root@network|compute:~# 命令
```

表示分别在网络节点和计算节点上执行“命令”。

```
root@all:~# 命令
```

表示在所有节点上执行“命令”，即同一个命令需要在三个节点上各执行一次。

1．准备基础环境

参照“8.2.1 系统设计”小节在 VMware WorkStation Pro 12 中创建三台虚拟机，然后分别安装 ubuntu-16.04-server-amd64。

（1）修改主机名称。

```
root@controller:~# hostnamecctl set-hostname controller
root@network:~# hostnamecctl set-hostname network
root@compute:~# hostnamecctl set-hostname compute

root@all:~# echo “10.6.60.45   controller” >>/etc/hosts
root@all:~# echo “10.6.60.46   network” >>/etc/hosts
root@all:~# echo “10.6.60.47   compute” >>/etc/hosts

root@all:~# shutdown -r now
```

（2）配置网络。

```
# 首先查看系统中被检测到的全部网卡名称。
root@all:~# dmesg | grep ens
```

```
    [    2.630776] e1000 0000:02:01.0 ens33: renamed from eth0
    [    2.616475] e1000 0000:02:02.0 ens34: renamed from eth1
    [    2.642861] e1000 0000:02:06.0 ens38: renamed from eth2

  # 查看每块网卡的MAC 地址，结合图8-9，根据MAC 地址确定网卡的对应关系，例如查看ens33 的MAC：
  root@all:~# ifconfig ens33 | grep HWaddr
    ens33     Link encap:Ethernet  HWaddr 00:0c:29:da:41:6b

  root@controller:~# vim /etc/network/interfaces
    source /etc/network/interfaces.d/*

    auto lo
    iface lo inet loopback

    # ens33 网卡接入管理网络：
    auto ens33
    iface ens33 inet static
    address 10.6.60.45
    gateway 10.6.60.1
    netmask 255.255.255.0
    dns-nameservers 202.96.128.166 8.8.8.8

    # ens38 网卡接入提供商网络，但不加入网桥，以后用来测试OpenStack 虚拟机的网络，如ping 或SSH 登录，所以要配置IP。
    auto ens34
    iface ens34 inet static
    address 10.7.70.45
    netmask 255.255.255.0


  root@network:~# vim /etc/network/interfaces
    source /etc/network/interfaces.d/*

    auto lo
    iface lo inet loopback

    # ens33 网卡接入管理网络：
    auto ens33
    iface ens33 inet static
    address 10.6.60.46
    gateway 10.6.60.1
    netmask 255.255.255.0
    dns-nameservers 202.96.128.166 8.8.8.8

    # ens38 网卡接入提供商网络，由于此网卡要加入网桥，所以不用配IP：
    auto ens34
    iface ens34 inet manual
```

```
    up ip link set dev ens34 up
    down ip link set dev ens34 down

    # ens39 网卡接入叠加网络:
    auto ens38
    iface ens38 inet static
    address 10.8.80.46
    netmask 255.255.255.0


  root@compute:~# vim /etc/network/interfaces
    source /etc/network/interfaces.d/*

    auto lo
    iface lo inet loopback

    # ens33 网卡接入管理网络:
    auto ens33
    iface ens33 inet static
    address 10.6.60.47
    gateway 10.6.60.1
    netmask 255.255.255.0
    dns-nameservers 202.96.128.166 8.8.8.8

    # ens38 网卡接入提供商网络，由于此网卡要加入网桥，所以不用配IP:
    auto ens34
    iface ens34 inet manual
    up ip link set dev ens34 up
    down ip link set dev ens34 down

    # ens39 网卡接入管理网络:
    auto ens38
    iface ens38 inet static
    address 10.8.80.47
    netmask 255.255.255.0

  root@all:~# systemctl restart networking

  root@all:~# ping -c 3 www.bing.com
    PING cn-0001.cn-msedge.net (202.89.233.100) 56(84) bytes of
data.
    64 bytes from 202.89.233.100: icmp_seq=1 ttl=110 time=43.0 ms
    64 bytes from 202.89.233.100: icmp_seq=2 ttl=110 time=43.0 ms
```

（3）升级系统软件到最新版。

```
  root@all:~# apt update -y && apt upgrade -y
  root@all:~# shutdown -r now
```

（4）安装时钟同步软件。

为了保证三个节点的系统时间一样，可以这样做：控制节点的时间与互联网上的机器同步，网络节点和计算节点与控制节点同步。

```
root@all:~# apt install chrony -y

# 控制节点允许局域网内的其他机器同步时间。
root@controller:~# vim /etc/chrony/chrony.conf
  …
  pool 2.debian.pool.ntp.org offline iburst
  …
  allow 10.6.60.0/24
  …

# 网络节点和计算节点的时间与控制节点同步。
root@network|compute:~# vim /etc/chrony/chrony.conf
  …
  #pool 2.debian.pool.ntp.org offline iburst
  server controller iburst
  …
root@all:~# systemctl restart chrony

# 查看每个节点的同步服务器（以^*开头的行）。
root@controller:~# chronyc sources
   210 Number of sources = 7
   MS Name/IP address         Stratum Poll Reach LastRx Last sample
   ===============================================================
   ^* 172.104.71.235  2   6   373   30   -47ms[  -45ms] +/-  110ms
   ^+ 69.60.114.223   2   6   377   109   +65ms[  +68ms] +/-  227ms
   ^+ 5.103.139.163   1   6   377   65   +60ms[  +61ms] +/-  245ms

root@compute|network:~# chronyc sources
   210 Number of sources = 1
   MS Name/IP address         Stratum Poll Reach LastRx Last sample
   ===============================================================
   ^* controller    3   6    17    37    -12us[ -300us] +/-  122ms
```

（5）安装数据库。

```
root@controller:~# apt install mariadb-server python-pymysql
root@controller:~#                                              vim
/etc/mysql/mariadb.conf.d/99-OpenStack.cnf
  [mysqld]
  bind-address = 10.6.60.45
  default-storage-engine = innodb
```

```
    innodb_file_per_table = on
    max_connections = 4096
    collation-server = utf8_general_ci
    character-set-server = utf8

root@controller:~# systemctl restart mysql
root@controller:~# mysql_secure_installation
    ……
    Enter current password for root (enter for none):
    ……
    Set root password? [Y/n] Y
    New password:MYSQL_PASS
    Re-enter new password:MYSQL_PASS
    Password updated successfully!
    ……

    Remove anonymous users? [Y/n] Y
     ... Success!

    ……
    Disallow root login remotely? [Y/n] n
     ... skipping.
    ……
    Remove test database and access to it? [Y/n] Y
    ……

    Reload privilege tables now? [Y/n] Y
     ... Success!

    All done!  If you've completed all of the above steps, your MariaDB
    installation should now be secure.

    Thanks for using MariaDB!
```

最后一条命令给MariaDB数据库的root用户设置密码为MYSQL_PASS。

（6）安装消息队列工具。

```
root@controller:~# apt install rabbitmq-server -y
#创建消息队列用户OpenStack，并设置密码RABBIT_PASS：
root@controller:~# rabbitmqctl add_user OpenStack RABBIT_PASS

#赋予用户OpenStack配置读、写任何资源的权限：
root@controller:~# rabbitmqctl set_permissions OpenStack ".*" ".*"
".*"

root@controller:~# rabbitmqctl list_permissions
    Listing permissions in vhost "/"
```

```
    guest   .*      .*      .*
    OpenStack       .*      .*      .*
```

（7）安装 Memcached 工具。

```
root@controller:~# apt install memcached python-memcache -y
root@controller:~# vim /etc/memcached.conf
  ……
  -l 10.6.60.45
  ……
root@controller:~# systemctl restart memcached
```

（8）设置 OpenStack 的安装源，并安装 OpenStack 客户端工具。

```
root@all:~# apt install software-properties-common
root@all:~# add-apt-repository cloud-archive:queens
  Ubuntu Cloud Archive for OpenStack Queens
  More info: https://wiki.ubuntu.com/ServerTeam/CloudArchive
  Press [ENTER] to continue or ctrl-c to cancel adding it  ←这里直接按 Enter 键。

  Reading package lists...
root@all:~# apt update && apt dist-upgrade -y
root@all:~# apt install python-openstackclient -y
root@all:~# shutdown -r now
```

建议现在为三台 VMware 虚拟机各创建一个快照，取名为 before_install_keystone，当出现问题时便回滚到现在这个点。创建快照的方法：关闭 VMware 虚拟机，然后右键单击虚拟机的名称，后续的操作，如图 8-10 所示。

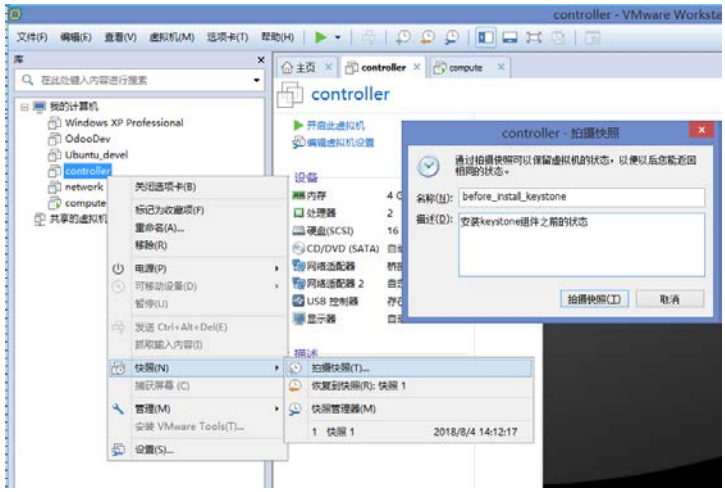


图 8-10　给 VMware 虚拟机创建快照

2．安装认证服务：Keystone

（1）建立数据库和 Keystone 用户。

```
   root@controller:~# mysql
   MariaDB [(none)]> CREATE DATABASE keystone;
   MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  keystone.*  TO
'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';
   MariaDB  [(none)]>  GRANT  ALL  PRIVILEGES  ON  keystone.*  TO
'keystone'@'%' IDENTIFIED BY 'KEYSTONE_DBPASS';
   MariaDB [(none)]> \q
```

（2）安装和配置相应的组件。

```
   root@controller:~#     apt     install     keystone     apache2
libapache2-mod-wsgi -y
   root@controller:~# vim /etc/keystone/keystone.conf
     # …
     [database]
     connection                                                    =
mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/keystone
     ……
     [token]
     # ...
     provider = fernet
```

向 Keystone 数据库导入数据。

```
   root@controller:~# su -s /bin/sh -c "keystone-manage db_sync"
keystone
```

初始化 fernet 密钥存储库。

```
   root@controller:~# keystone-manage fernet_setup --keystone-user
keystone --keystone-group keystone
   root@controller:~#       keystone-manage       credential_setup
--keystone-user keystone --keystone-group keystone
```

初始化身份认证服务。

```
   root@controller:~# keystone-manage bootstrap --bootstrap-password
ADMIN_PASS \
     --bootstrap-admin-url http://controller:5000/v3/ \
     --bootstrap-internal-url http://controller:5000/v3/ \
     --bootstrap-public-url http://controller:5000/v3/ \
     --bootstrap-region-id RegionOne
```

配置 Apache。

```
   root@controller:~# vim /etc/apache2/apache2.conf
```

```
    # …
    ServerName controller
    ……
root@controller:~# systemctl restart apache2
```

配置 Keystone 管理员的环境变量。

```
root@controller:~# export OS_USERNAME=admin
root@controller:~# export OS_PASSWORD=ADMIN_PASS
root@controller:~# export OS_PROJECT_NAME=admin
root@controller:~# export OS_USER_DOMAIN_NAME=Default
root@controller:~# export OS_PROJECT_DOMAIN_NAME=Default
root@controller:~# export OS_AUTH_URL=http://controller:5000/v3
root@controller:~# export OS_IDENTITY_API_VERSION=3
```

（3）创建域、项目、用户和角色。

在 Keystone 中默认存在 default 域，允许创建更多的域，比如下面的命令创建 example 域。

```
root@controller:~# openstack domain create --description "An
Example Domain" example
   +-------------+----------------------------------+
   | Field       | Value                            |
   +-------------+----------------------------------+
   | description | An Example Domain                |
   | enabled     | True                             |
   | id          | f22b113b07d6489094a2f09e95c7cf2b |
   | name        | example                          |
   | tags        | []                               |
   +-------------+----------------------------------+
```

在 default 域中创建 service 项目用于存放全部的服务。

```
root@controller:~# openstack project create --domain default
--description "Service Project" service
```

创建一个普通项目 myProject 和一个普通用户 metoo，用于完成普通的任务。

```
root@controller:~# openstack project create --domain default
--description "Test Project" myProject
root@controller:~# openstack user create --domain default
--password-prompt metoo
   User Password:METOO_PASS
   Repeat User Password:METOO_PASS
root@controller:~# openstack role create user
root@controller:~# openstack role add --project myProject --user
metoo user
```

最后一条命令完成授权：即在 myProject 项目中授予 metoo 用户 user 角色，user 角色具备普通用户权限。

（4）验证 Keystone。

先删除几个环境变量。

```
root@controller:~# unset OS_AUTH_URL OS_PASSWORD
```

获取 keystone 管理员的令牌。

```
root@controller:~#           openstack           --os-auth-url
http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name admin --os-username admin token issue
  Password:ADMIN_PASS
```

输入用户 admin 的密码 ADMIN_PASS，如果一切正常就会获得一个令牌。

获取 Keystone 普通用户 metoo 的令牌。

```
root@controller:~#           openstack           --os-auth-url
http://controller:5000/v3 \
  --os-project-domain-name Default --os-user-domain-name Default \
  --os-project-name myProject --os-username metoo token issue
  Password:METOO_PASS
```

输入用户 metoo 的密码 METOO_PASS，如果一切正常就会获得一个令牌。

（5）创建生成环境变量的脚本文件。

产生 admin 用户的环境变量的脚本文件 admin-openrc。

```
root@controller:~# vim admin-openrc
  export OS_PROJECT_DOMAIN_NAME=Default
  export OS_USER_DOMAIN_NAME=Default
  export OS_PROJECT_NAME=admin
  export OS_USERNAME=admin
  export OS_PASSWORD=ADMIN_PASS
  export OS_AUTH_URL=http://controller:5000/v3
  export OS_IDENTITY_API_VERSION=3
  export OS_IMAGE_API_VERSION=2
```

产生 metoo 用户的环境变量的脚本文件 metoo-openrc。

```
root@controller:~# vim metoo-openrc
  export OS_PROJECT_DOMAIN_NAME=Default
  export OS_USER_DOMAIN_NAME=Default
  export OS_PROJECT_NAME=myProject
  export OS_USERNAME=metoo
  export OS_PASSWORD=METOO_PASS
  export OS_AUTH_URL=http://controller:5000/v3
  export OS_IDENTITY_API_VERSION=3
  export OS_IMAGE_API_VERSION=2
```

以后就可以“点”执行这些文件来快速产生环境变量，如下所示。

```
root@controller:~# . admin-openrc
root@controller:~# openstack token issue
```

3．安装镜像服务：Glance

（1）创建数据库、数据库用户并授权。

```
root@controller:~# mysql -uroot -pMYSQL_PASS
MariaDB [(none)]> CREATE DATABASE glance;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO
'glance'@'localhost' IDENTIFIED BY 'GLANCE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%'
IDENTIFIED BY 'GLANCE_DBPASS';
MariaDB [(none)]> \q
```

（2）建立 keystone 用户并授权。

```
# “点”执行脚本程序admin-openrc，从而输出里面定义的环境变量。
root@controller:~# . admin-openrc

# 在keystone中创建用户glance，密码为GLANCE_PASS。
root@controller:~#  openstack  user  create  --domain  default
--password-prompt glance
  User Password:GLANCE_PASS
  Repeat User Password:GLANCE_PASS

# 在service项目中给用户glance授予管理员权限admin。
root@controller:~# openstack role add --project service --user
glance admin

# 创建服务类型为image的实体 glance。
root@controller:~#  openstack  service  create  --name  glance
--description "OpenStack Image" image
```

（3）创建镜像服务的调用端点。

```
# 为image服务类型创建公开（public）的调用端点。
root@controller:~# openstack endpoint create --region RegionOne
image public http://controller:9292
# 为image服务类型创建内部（internal）的调用端点。
root@controller:~# openstack endpoint create --region RegionOne
```

```
image internal http://controller:9292
   # 为image服务类型创建管理（admin）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
image admin http://controller:9292
```

根据权限的不同，OpenStack 把调用端点（endpoint）分为三类，分别是public、internal 和 admin。

（4）安装并配置相关软件。

```
   root@controller:~# apt install glance -y
   root@controller:~# vim /etc/glance/glance-api.conf

     [database]
     # 配置glance服务如何访问数据库。
     connection                                                    =
mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
     #connection = sqlite:////var/lib/glance/glance.sqlite
     # backend = sqlalchemy

     [glance_store]
     # 配置镜像文件格式和存放位置：普通文件(file)，存放在
/var/lib/glance/images。
     stores = file,http
     default_store = file
     filesystem_store_datadir = /var/lib/glance/images/

     [keystone_authtoken]
     # glance用户如何访问keystone。
     auth_uri = http://controller:5000
     auth_url = http://controller:5000
     memcached_servers = controller:11211
     auth_type = password
     project_domain_name = Default
     user_domain_name = Default
     project_name = service
     username = glance
     password = GLANCE_PASS

     [paste_deploy]
     flavor = keystone
```

修改文件/etc/glance/glance-api.conf 时要特别注意：上面列出的节[database]、[glance_store]、[keystone_authtoken]和[paste_deploy]中其他的行要添加注释，不要修改文件中的其他节。

```
   root@controller:~# vim /etc/glance/glance-registry.conf

     [database]
     # 配置glance服务如何访问数据库。
     connection                                                    =
```

```
mysql+pymysql://glance:GLANCE_DBPASS@controller/glance

     [keystone_authtoken]
     # glance 用户如何访问 keystone。
     auth_uri = http://controller:5000
     auth_url = http://controller:5000
     memcached_servers = controller:11211
     auth_type = password
     project_domain_name = Default
     user_domain_name = Default
     project_name = service
     username = glance
     password = GLANCE_PASS

     [paste_deploy]
     flavor = keystone
```

上面列出的节[database]、[keystone_authtoken]和[paste_deploy]中其他的行要添加注释，不要修改文件中的其他节。

（5）向数据库中导入 Glance 的数据。

```
   root@controller:~# su -s /bin/sh -c "glance-manage db_sync" glance
```

（6）启动 Glance 服务。

```
   root@controller:~# systemctl restart glance-api
   root@controller:~# systemctl restart glance-registry
```

（7）验证。

```
   root@controller:~# . admin-openrc
   # 把文件 cirros-0.4.0-x86_64-disk.img 下载到当前目录中。
   root@controller:~# wget
http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.im
g
   # 使用刚刚下载的文件创建镜像 cirros。
   root@controller:~# openstack image create "cirros" --file
cirros-0.4.0-x86_64-disk.img --disk-format qcow2 --container-format
bare --public
   root@controller:~# openstack image list
     +--------------------------------------+--------+--------+
     | ID                                   | Name   | Status |
     +--------------------------------------+--------+--------+
     | 18a78d61-84c6-4c39-8e7f-fda4682c14e6 | cirros | active |
     +--------------------------------------+--------+--------+
```

如果看到上面最后一条命令的输出，就说明 Glance 服务安装成功。

4．安装计算服务：Nova

以下在控制节点上操作。

（1）创建数据库、数据库用户并授权。

```
   root@controller:~# mysql -uroot -pMYSQL_PASS
   MariaDB [(none)]> CREATE DATABASE nova_api;
   MariaDB [(none)]> CREATE DATABASE nova;
   MariaDB [(none)]> CREATE DATABASE nova_cell0;
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_DBPASS';
   MariaDB [(none)]> \q
```

（2）建立 Keystone 用户并授权。

```
   # “点”执行脚本程序 admin-openrc，从而输出里面定义的环境变量。如果没有注销
Linux 的用户，环境变量一直存在，也可以不“点”执行。
   root@controller:~# . admin-openrc

   # 在 keystone 中创建用户 nova，密码为 NOVA_PASS。
   root@controller:~# openstack user create --domain default
--password-prompt nova
     User Password:NOVA_PASS
     Repeat User Password:NOVA_PASS

   # 在 service 项目中给用户 nova 授予管理员权限 admin。
   root@controller:~# openstack role add --project service --user nova
admin

   # 创建服务类型为 compute 的实体 nova。
   root@controller:~# openstack service create --name nova
--description "OpenStack Compute" compute
```

（3）创建计算服务的调用端点。

```
   # 为 compute 服务类型创建公开（public）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
compute public http://controller:8774/v2.1
   # 为 compute 服务类型创建内部（internal）的调用端点。
```

```
   root@controller:~# openstack endpoint create --region RegionOne
compute internal http://controller:8774/v2.1
   # 为compute 服务类型创建管理（admin）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
compute admin http://controller:8774/v2.1
```

（4）建立 placement 用户并授权。

```
   # 在keystone 中创建用户placement，密码为PLACEMENT_PASS。
   root@controller:~#  openstack  user  create  --domain  default
--password-prompt placement
     User Password:PALCEMENT_PASS
     Repeat User Password:PLACEMENT_PASS
   # 在service 项目中给用户placement 授予管理员权限admin。
   root@controller:~# openstack role add --project service --user
placement admin
   # 创建服务类型为placement 的实体 placement。
   root@controller:~# openstack service create --name placement
--description "Placement API" placement
```

（5）创建相应的调用端点。

```
   # 为placement 服务类型创建公开（public）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
placement public http://controller:8778
   # 为placement 服务类型创建内部（internal）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
placement internal http://controller:8778
   # 为placement 服务类型创建管理（admin）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
placement admin http://controller:8778
```

（6）安装并配置相关软件。

```
   root@controller:~# apt install nova-api nova-conductor
nova-consoleauth nova-novncproxy nova-scheduler nova-placement-api -y

   root@controller:~# vim /etc/nova/nova.conf
     [DEFAULT]
     lock_path = /var/lock/nova
     state_path = /var/lib/nova
     transport_url = rabbit://OpenStack:RABBIT_PASS@controller
     my_ip = 10.6.60.45
     use_neutron = True
     firewall_driver = nova.virt.firewall.NoopFirewallDriver

     [api]
     auth_strategy = keystone

     [api_database]
     connection                                                    =
mysql+pymysql://nova:NOVA_DBPASS@controller/nova_api
     # connection = sqlite:////var/lib/nova/nova_api.sqlite
```

```
[database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova
# connection = sqlite:////var/lib/nova/nova.sqlite

[glance]
api_servers = http://controller:9292

[keystone_authtoken]
auth_url = http://controller:5000/v3
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = nova
password = NOVA_PASS

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
username = placement
password = PLACEMENT_PASS

[vnc]
enabled = true
server_listen = $my_ip
server_proxyclient_address = $my_ip
```

修改文件/etc/nova/nova.conf 时需要注意：上面列出的节中其他的行要全部添加注释或删除，文件中的其他节不要改动。

（7）向数据库中导入 nova、nova_api、nova_cell0 相关的数据。

```
root@controller:~# su -s /bin/sh -c "nova-manage api_db sync" nova
root@controller:~# su -s /bin/sh -c "nova-manage cell_v2 map_cell0"
nova
root@controller:~# su -s /bin/sh -c "nova-manage cell_v2
create_cell --name=cell1 --verbose" nova
root@controller:~# su -s /bin/sh -c "nova-manage db sync" nova
```

忽略任何有关“过期参数”的警告信息。

（8）检查 cell0 和 cell1 是否注册成功。

```
root@controller:~# nova-manage cell_v2 list_cells
  +-------+--------------------------------------+----------------
```

```
|  Name |              UUID                  | Transport URL ...
+-------+------------------------------------+----------------
|cell0 |00000000-0000-0000-0000-000000000000|
| cell1 | a9b6555a-2714-4d78-b607-c06957a87299 | rabbit://opens...
+-------+------------------------------------+----------------
```

（9）重启控制节点上的所有 Nova 服务。

```
root@controller:~# systemctl stop nova-api
root@controller:~# systemctl stop nova-consoleauth
root@controller:~# systemctl stop nova-scheduler
root@controller:~# systemctl stop nova-conductor
root@controller:~# systemctl stop nova-novncproxy

root@controller:~# rm /var/log/nova/*

root@controller:~# systemctl start nova-api
root@controller:~# systemctl start nova-consoleauth
root@controller:~# systemctl start nova-scheduler
root@controller:~# systemctl start nova-conductor
root@controller:~# systemctl start nova-novncproxy

root@controller:~# grep ERROR /var/log/nova/*
```

如果最后一条命令有输出，说明某些服务遇到问题，需要处理完问题之后再往下操作。

以下操作在计算节点上进行。

（10）安装并配置计算软件包。

```
root@compute:~# apt install nova-compute -y

root@compute:~# vim /etc/nova/nova.conf
  [DEFAULT]
  lock_path = /var/lock/nova
  state_path = /var/lib/nova
  transport_url = rabbit://OpenStack:RABBIT_PASS@controller
  my_ip = 10.6.60.47
  use_neutron = True
  firewall_driver = nova.virt.firewall.NoopFirewallDriver

  [api]
  auth_strategy = keystone

  [glance]
  api_servers = http://controller:9292

  [keystone_authtoken]
  auth_url = http://controller:5000/v3
  memcached_servers = controller:11211
  auth_type = password
  project_domain_name = default
```

```
user_domain_name = default
project_name = service
username = nova
password = NOVA_PASS

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
username = placement
password = PLACEMENT_PASS

[vnc]
enabled = true
server_listen = 0.0.0.0
server_proxyclient_address = $my_ip
novncproxy_base_url = http://controller:6080/vnc_auto.html
```

上面列出的节中其他的行要全部添加注释或删除，文件中的其他节不要改动。

（11）检查机器的 CPU 是否支持虚拟化。

```
root@compute:~# egrep -c '(vmx|svm)' /proc/cpuinfo
```

（12）重启 nova-compute 服务。

```
root@compute:~# systemctl stop nova-compute
root@compute:~# rm /var/log/nova/*
root@compute:~# systemctl start nova-compute
root@compute:~# grep ERROR /var/log/nova/*
```

如果看到上面最后一条命令有输出，说明 Nova 的某些服务存在问题，需要根据具体的日志排查。最常见的错误是 nova-compute 服务与消息队列连接失败。

以下操作在控制节点上进行。

（13）把计算节点添加到 cell 数据库。

```
root@controller:~# . admin-openrc

# 检查计算节点上的 nova-compute 服务是否运行正常，如果状态为 down，需要检查原因，一般重启机器会解决。
```

```
   root@controller:~#  openstack  compute  service  list  --service
nova-compute
     +----+--------------+--------+------+---------+-------+------
----------------------+
     | ID | Binary       | Host   | Zone | Status  | State | Updated At
|
     +----+--------------+--------+------+---------+-------+------
----------------------+
     |  8  |  nova-compute  |  compute|  nova  |  enabled  |  up    |
2018-07-26T08:11:01.000000 |
     +----+--------------+--------+------+---------+-------+------
----------------------+

   root@controller:~#  su  -s  /bin/sh  -c  "nova-manage  cell_v2
discover_hosts --verbose" nova
      Found 2 cell mappings.
      Skipping cell0 since it does not contain hosts.
      Getting       computes       from       cell       'cell1':
a9b6555a-2714-4d78-b607-c06957a87299
      Checking   host   mapping   for   compute   host   'compute':
95fc30f1-3733-4eaf-8df5-b5f133729b63
      Creating   host   mapping   for   compute   host   'compute':
95fc30f1-3733-4eaf-8df5-b5f133729b63
      Found      1      unmapped      computes      in      cell:
a9b6555a-2714-4d78-b607-c06957a87299
```

（14）验证 Nova 安装是否成功。

```
   root@controller:~# . admin-openrc

   # 查看计算相关的服务启动是否正常。
   root@controller:~# openstack compute service list
   +----+------------------+------------+----------+---------+----
-+--------------------+
   | ID | Binary           | Host       | Zone     | Status  | Stat|
Updated At        |
   +----+------------------+------------+----------+---------+----
-+--------------------+
   |  1 | nova-scheduler   | controller | internal | enabled | up  |
2018-07-26T08:18:08|
   |  5 | nova-consoleauth | controller | internal | enabled | up  |
2018-07-26T08:18:06|
   |  6 | nova-conductor   | controller | internal | enabled | up  |
2018-07-26T08:18:14|
   |  8 | nova-compute     | compute    | nova     | enabled | up  |
2018-07-26T08:18:11|
   +----+------------------+------------+----------+---------+----
-+--------------------+
```

```
# 分类查看全部的调用端点。
root@controller:~# openstack catalog list
   +---------+--------+-----------------------------------+
   | Name    | Type   | Endpoints                         |
   +---------+--------+-----------------------------------+
   | glance  | image  | RegionOne                         |
   |         |        |  internal: http://controller:9292 |
   |         |        | RegionOne                         |
   |         |        |  public: http://controller:9292   |
   |         |        | RegionOne                         |
   |         |        |  admin: http://controller:9292    |
   |         |        |                                   |
   | keystone|identity| RegionOne                         |
   |         |        |  internal: http://controller:5000/v3/ |
   |         |        | RegionOne                         |
   |         |        |  admin: http://controller:5000/v3/ |
   |         |        | RegionOne                         |
   |         |        |  public: http://controller:5000/v3/ |
   |         |        |                                   |
   | nova    | compute | RegionOne                        |
   |         |        |  admin: http://controller:8774/v2.1 |
   |         |        | RegionOne                         |
   |         |        |  public: http://controller:8774/v2.1 |
   |         |        | RegionOne                         |
   |         |        |  internal: http://controller:8774/v2.1 |
   |         |        |                                   |
   |placement|placement| RegionOne                        |
   |         |        |  internal: http://controller:8778 |
   |         |        | RegionOne                         |
   |         |        |  public: http://controller:8778   |
   |         |        | RegionOne                         |
   |         |        |  admin: http://controller:8778    |
   |         |        |                                   |
   +-----------+----------+---------------------------------+

# 检查cell和placement API是否正常工作。
root@controller:~# nova-status upgrade check
   +--------------------------------+
   | Upgrade Check Results          |
   +--------------------------------+
   | Check: Cells v2                |
   | Result: Success                |
   | Details: None                  |
   +--------------------------------+
   | Check: Placement API           |
   | Result: Success                |
   | Details: None                  |
   +--------------------------------+
```

```
      | Check: Resource Providers      |
      | Result: Success                |
      | Details: None                   |
      +--------------------------------+
      | Check: Ironic Flavor Migration |
      | Result: Success                |
      | Details: None                  |
      +--------------------------------+
      | Check: API Service Version     |
      | Result: Success                |
      | Details: None                   |
      +--------------------------------+

   # 检查计算节点上是否创建了virbr0桥，virbr0是安装KVM时自动创建的桥。
   root@compute:~# brctl show
      bridge name     bridge id             STP enabled     interfaces
      virbr0          8000.5254002d7bae      yes
virbr0-nic
```

如果最后那条命令返回警告信息：There are no compute resource providers in the Placement, service but there are 1 compute nodes in the deployment…。重点检查/etc/nova/nova.conf 中的[keystone_authtoken]节中的参数是否正确。

现在再为控制节点和计算节点各创建一个快照，取名为 before_install_neutron，这样在部署 Neutron 的过程中可以随时回滚 VMware 虚拟机到现在这个安装点。

5．安装网络服务：Neutron

Neutron 是最复杂的 OpenStack 组件，下面遵循由易到难的原则一步一步地完成任务，即从 Provider 网络开始，然后配置成 self-service 网络，最后启用 DVR。这次我们采用 Open vSwitch，而不是 LinuxBridge。

1）Provider 网络

提供商网络为虚拟机和物理局域网之间提供二层连通性，支持一个普通局域网和 4095 个 VLAN。也就是提供商网络就是云端物理机所在的局域网(一般这个局域网再通过路由器连接因特网)，OpenStack 虚拟机的网卡 IP 地址就设在这个局域网内。提供商网络没有用到网络节点，所以在本次实验中我

们不需要配置网络节点，如图 8-11 所示。

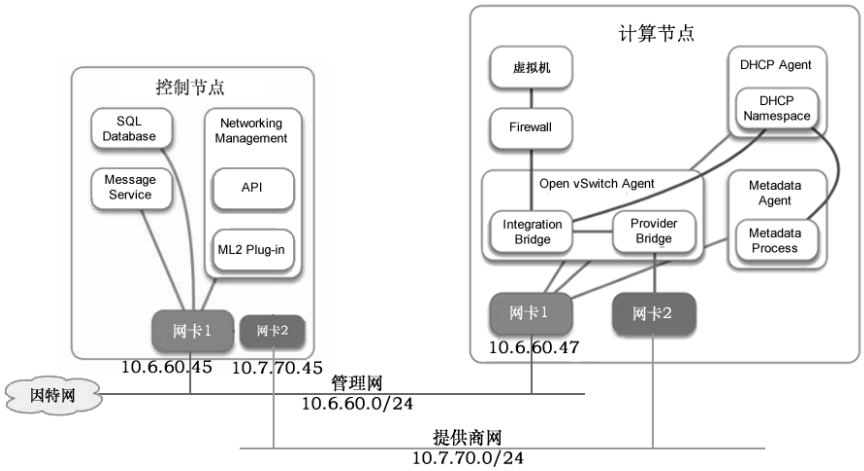


图 8-11　Provider 网络

以下在控制节点上操作。

（1）创建数据库、数据库用户并授权。

```
   root@controller:~# mysql -uroot -pMYSQL_PASS
   MariaDB [(none)]> CREATE DATABASE neutron;
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO
'neutron'@'localhost' IDENTIFIED BY 'NEUTRON_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%'
IDENTIFIED BY 'NEUTRON_DBPASS';
   MariaDB [(none)]> \q
```

（2）创建 Keystone 用户并授权。

```
   # “点”执行脚本程序admin-openrc，从而输出里面定义的环境变量。
   root@controller:~# . admin-openrc
   # 在keystone 中创建用户neutron，密码为NEUTRON_PASS。
   root@controller:~#  openstack  user  create  --domain  default
--password-prompt neutron
     User Password:NEUTRON_PASS
     Repeat User Password:NEUTRON_PASS
   # 在service 项目中给用户neutron 授予管理员权限admin。
   root@controller:~# openstack role add --project service --user
neutron admin
   # 创建服务类型为network 的实体 neutron。
   root@controller:~# openstack service create --name neutron
--description "OpenStack Networking" network
```

（3）创建镜像服务的调用端点。

```
   # 为network 服务类型创建公开（public）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
network public http://controller:9696
   # 为network 服务类型创建内部（internal）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
network internal http://controller:9696
   # 为network 服务类型创建管理（admin）的调用端点。
   root@controller:~# openstack endpoint create --region RegionOne
network admin http://controller:9696
```

（4）安装并配置相关软件。

```
   root@controller:~#     apt     -y     install     neutron-server
neutron-plugin-ml2 python-neutronclient

   root@controller:~# vim /etc/neutron/neutron.conf

     [DEFAULT]
     core_plugin = ml2
     service_plugins =
     transport_url = rabbit://OpenStack:RABBIT_PASS@controller
     auth_strategy = keystone
     notify_nova_on_port_status_changes = true
     notify_nova_on_port_data_changes = true

     [database]
     # 配置neutron 服务如何访问数据库。
     connection                                                    =
mysql+pymysql://neutron:NEUTRON_DBPASS@controller/neutron
     # connection = sqlite:////var/lib/neutron/neutron.sqlite

     [keystone_authtoken]
     # neutron 用户如何访问keystone。
     auth_uri = http://controller:5000
     auth_url = http://controller:5000
     memcached_servers = controller:11211
     auth_type = password
     project_domain_name = default
     user_domain_name = default
     project_name = service
     username = neutron
     password = NEUTRON_PASS

     [nova]
     auth_url = http://controller:5000
     auth_type = password
```

```
    project_domain_name = default
    user_domain_name = default
    region_name = RegionOne
    project_name = service
    username = nova
    password = NOVA_PASS
```

修改文件/etc/neutron/neutron.conf 时要特别注意：上面列出的节[DEFAULT]、[database]、[keystone_authtoken]、[nova]中其他的行要添加注释，不要修改文件中的其他节。

```
root@controller:~# vim /etc/neutron/plugins/ml2/ml2_conf.ini

    [ml2]
    type_drivers = flat,vlan
    tenant_network_types =
    # mechanism_drivers = linuxbridge,l2population
    mechanism_drivers = openvswitch
    extension_drivers = port_security

    [ml2_type_flat]
    flat_networks = my_physical_lan

    [ml2_type_vlan]
    network_vlan_ranges = my_physical_lan:10:1000
```

上面列出的节[ml2]、[ml2_type_flat]、[ml2_type_vlan]中其他的行要添加注释，不要修改文件中的其他节。

配置计算服务访问网络服务。

```
root@controller:~# vim /etc/nova/nova.conf

    [DEFAULT]
    …
    #                 linuxnet_interface_driver                 =
nova.network.linux_net.LinuxBridgeInterfaceDriver
    linuxnet_interface_driver                                   =
nova.network.linux_net.LinuxOVSInterfaceDriver

    [neutron]
    url = http://controller:9696
    auth_url = http://controller:5000
    auth_type = password
    project_domain_name = default
```

```
    user_domain_name = default
    region_name = RegionOne
    project_name = service
    username = neutron
    password = NEUTRON_PASS
    service_metadata_proxy = true
    metadata_proxy_shared_secret = METADATA_PASS
```

上面的“...”表示[DEFAULT]节中原有的内容不动，只是增加如下一行：

```
   linuxnet_interface_driver       =       nova.network.linux_net.
LinuxOVSInterfaceDriver
```

文件/etc/nova/nova.conf 中其他的节不要修改。

（5）向数据库中导入 Neutron 的数据。

```
   root@controller:~# su -s /bin/sh -c "neutron-db-manage
--config-file /etc/neutron/neutron.conf --config-file
/etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron
```

（6）启动有关服务。

```
   root@controller:~# systemctl restart nova-api

   root@controller:~# systemctl stop neutron-server
   root@controller:~# rm /var/log/neutron/*
   root@controller:~# systemctl start neutron-server
   root@controller:~# grep ERROR /var/log/neutron/*
```

如果看到上面倒数第一条命令有输出，说明 Neutron 服务安装异常，可以再查看 9696 端口是否被监听，作进一步确认。

```
   root@controller:~# netstat -tnlp|grep 9696
     tcp   0  0 0.0.0.0:9696   0.0.0.0:*    LISTEN     22911/python2
```

以下在计算节点上操作。

（7）安装和配置 Neutron 有关软件。

```
   root@compute:~# apt-get -y install neutron-openvswitch-agent
neutron-plugin-ml2  neutron-metadata-agent neutron-dhcp-agent
openvswitch-switch

   root@compute:~# vim /etc/neutron/neutron.conf
     [DEFAULT]
     core_plugin = ml2
     transport_url = rabbit://OpenStack:RABBIT_PASS@controller
```

```
     auth_strategy = keystone

     [database]
     connection =
mysql+pymysql://neutron:NEUTRON_DBPASS@controller/neutron
     # connection = sqlite:////var/lib/neutron/neutron.sqlite

     [keystone_authtoken]
     auth_uri = http://controller:5000
     auth_url = http://controller:5000
     memcached_servers = controller:11211
     auth_type = password
     project_domain_name = default
     user_domain_name = default
     project_name = service
     username = neutron
     password = NEUTRON_PASS

     [nova]
     auth_url = http://controller:5000
     auth_type = password
     project_domain_name = default
     user_domain_name = default
     region_name = RegionOne
     project_name = service
     username = nova
     password = NOVA_PASS

   root@compute:~# vim /etc/neutron/dhcp_agent.ini
     [DEFAULT]
     interface_driver = openvswitch
     enable_isolated_metadata = True
     force_metadata = True

   root@compute:~# vim
/etc/neutron/plugins/ml2/openvswitch_agent.ini
     [ovs]
     bridge_mappings = my_physical_lan:br-provider

     [securitygroup]
     firewall_driver = iptables_hybrid

   root@compute:~# vim /etc/neutron/metadata_agent.ini
     [DEFAULT]
     nova_metadata_host = controller
     metadata_proxy_shared_secret = METADATA_PASS

     [cache]
```

```
    memcache_servers = controller:11211

  root@compute:~# vim /etc/neutron/plugins/ml2/ml2_conf.ini
    [ml2]
    type_drivers = flat,vlan
    tenant_network_types =
    # mechanism_drivers = linuxbridge,l2population
    mechanism_drivers = openvswitch
    extension_drivers = port_security

    [ml2_type_flat]
    flat_networks = my_physical_lan


  root@compute:~# vim /etc/nova/nova.conf

    [DEFAULT]
    …
    #                 linuxnet_interface_driver                  =
nova.network.linux_net.LinuxBridgeInterfaceDriver
    linuxnet_interface_driver                                    =
nova.network.linux_net.LinuxOVSInterfaceDriver

    [neutron]
    url = http://controller:9696
    auth_url = http://controller:5000
    auth_type = password
    project_domain_name = default
    user_domain_name = default
    region_name = RegionOne
    project_name = service
    username = neutron
    password = NEUTRON_PASS
    service_metadata_proxy = true
    metadata_proxy_shared_secret = METADATA_PASS
```

（8）创建网桥 br-provider。

```
  root@compute:~# systemctl restart openvswitch-switch

  root@compute:~# ovs-vsctl add-br br-provider

  # 把计算节点上连接提供商网络的网卡 ens34 加入此桥。
  root@compute:~# ovs-vsctl add-port br-provider ens34
  root@compute:~# ovs-vsctl show
     f2e64dfd-f7f0-4774-aaf3-35ddadffed25
        Manager "ptcp:6640:127.0.0.1"
           is_connected: true
```

```
        Bridge br-provider
            Port "ens34"
                Interface "ens34"
            Port br-provider
                Interface br-provider
                    type: internal
        Bridge br-int
            Controller "tcp:127.0.0.1:6633"
                is_connected: true
            fail_mode: secure
            Port br-int
                Interface br-int
                    type: internal
        ovs_version: "2.9.0"
```

（9）重启服务。

```
   root@compute:~# systemctl stop neutron-openvswitch-agent
neutron-dhcp-agent neutron-metadata-agent
   root@compute:~# rm /var/log/neutron/*
   root@compute:~# systemctl start neutron-openvswitch-agent
neutron-dhcp-agent neutron-metadata-agent
   root@compute:~# grep ERROR /var/log/neutron/*
```

（10）验证。

```
   root@controller:~# . admin-openrc
   root@controller:~# openstack network agent list
       +----+--------------------+---------+----------+-----+-----
+---------------------------+
       | ID | Agent Type         | Host    | Availabil| Aliv| Stat|
Binary                    |
       +----+--------------------+---------+----------+-----+-----
+---------------------------+
       | 2ac| Open vSwitch agent | compute | None     | :-) | UP  |
neutron-openvswitch-agent |
       | 6cf| Metadata agent     | compute | None     | :-) | UP  |
neutron-metadata-agent    |
       | 8c1| DHCP agent         | compute | nova     | :-) | UP  |
neutron-dhcp-agent        |
       +----+--------------------+---------+----------+-----+-----
+---------------------------+

   root@compute:~# ovs-vsctl show
      f2e64dfd-f7f0-4774-aaf3-35ddadffed25
         Manager "ptcp:6640:127.0.0.1"
            is_connected: true
         Bridge br-provider
            Controller "tcp:127.0.0.1:6633"
```

```
            is_connected: true
        fail_mode: secure
        Port "ens34"
            Interface "ens34"
        Port br-provider
            Interface br-provider
                type: internal
        Port phy-br-provider
            Interface phy-br-provider
                type: patch
                options: {peer=int-br-provider}
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
        Port int-br-provider
            Interface int-br-provider
                type: patch
                options: {peer=phy-br-provider}
    ovs_version: "2.9.0"
```

发现重启 neutron-openvswitch-agent 服务后，自动建立了 br-int 和 br-provider 两个桥的连接。

到此，一个最小的提供商网络型云端就安装好了，下面我们来创建并启动一台 OpenStack 虚拟机，全部在控制节点上操作。

（11）创建虚拟机。

```
    root@controller:~# . admin-openrc

    # 创建一个虚拟机资源框架。
    root@controller:~# openstack flavor create --id 1 --vcpus 1 --ram
64 --disk 1 m1.nano

    # 在提供商物理网my_physical_lan上创建flat型网络provider-net1。
    root@controller:~# openstack network create --share
--provider-physical-network my_physical_lan --provider-network-type
flat provider-net1

    # 创建子网provider-subnet1，分配的IP地址属于提供商网络。
    root@controller:~# openstack subnet create --subnet-range
10.7.70.0/24 --gateway 10.7.70.1 --network provider-net1
--allocation-pool start=10.7.70.11,end=10.7.70.20 --dns-nameserver
```

```
202.96.128.166 --dns-nameserver 8.8.4.4 provider-subnet1

   # 在每台计算节点上应该存在相同的网络名字空间（如果没有，就重启全部节点）。
   root@compute:~# ip netns
     qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539

   # 进一步查看此名字空间的网络设备参数。
   root@compute:~# ip netns exec
qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539 ifconfig
     …
     tap5b08aaf5-0b Link encap:Ethernet  HWaddr fa:16:3e:fb:47:c4
             inet addr:10.7.70.11  Bcast:10.7.70.255
Mask:255.255.255.0
             inet6 addr: fe80::f816:3eff:fefb:47c4/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     …
   # 发现有一块虚拟网卡tap5b08aaf5-0b被分配了IP地址10.7.70.11，显然这个
网络名字空间可以与下文中创建的虚拟机通信。进一步运行命令ovs-vsctl show，会发
现此名字空间通过虚拟网卡tap5b08aaf5-0b连接br-int桥。建议现在给控制节点和计
算节点创建快照，取名after_create_subnet。

   root@controller:~# . metoo-openrc

   # 在default默认安全组中开启ping和22号端口。
   root@controller:~# openstack security group rule create --proto
icmp default
   root@controller:~# openstack security group rule create --proto tcp
--dst-port 22 default

   # 查看网络provider-net1的ID号和flavor的ID号，创建虚拟机时要用到。
   root@controller:~# openstack network list
       +--------------------------------------+--------------+---
-----------------------------------+
       | ID                                   | Name         | Subnets      |
       +--------------------------------------+--------------+---
-----------------------------------+
       | e8d373b4-1308-43d3-8e01-fca5f41f2539 | provider-net1 |
49407d5a-06e1-47ce-aede-745e9e6ae135 |
       +--------------------------------------+--------------+---
-----------------------------------+
   root@controller:~# openstack flavor list
      +----+---------+-----+------+-----------+-------+----------+
      | ID | Name    | RAM | Disk | Ephemeral | VCPUs | Is Public |
      +----+---------+-----+------+-----------+-------+----------+
      | 1  | m1.nano |  64 |    1 |         0 |     1 | True     |
      +----+---------+-----+------+-----------+-------+----------+
   root@controller:~# openstack image list
      +--------------------------------------+--------+--------+
```

```
      | ID                                   | Name   | Status |
      +--------------------------------------+--------+--------+
      | dceddf12-8f5f-4730-a325-95514335dd91 | cirros | active |
      +--------------------------------------+--------+--------+

   # 创建虚拟机 provider-vm1 并启动它（网络 ID 号就是前面的命令查出来的）。
   root@contoller:~# openstack server create --flavor 1 --image cirros
--nic net-id=e8d373b4-1308-43d3-8e01-fca5f41f2539 provider-vm1

   # 检查虚拟机的状态和分配的 IP 地址。
   root@contoller:~# openstack server list
      +---------+--------------+--------+------------------------
-+--------+---------+
      | ID      | Name         | Status | Networks               | Image
| Flavor  |
      +---------+--------------+--------+------------------------
-+--------+---------+
      | 9ea49… | provider-vm1 | ACTIVE | provider-net1=10.7.70.19 |
cirros | m1.nano |
      +---------+--------------+--------+------------------------
-+--------+---------+
```

我们不能在计算节点上直接 ping 通虚拟机的 IP，因为计算节点上接到提供商网的网卡被添加到桥中，从而作为桥的 trunk 口了。此时可以在控制节点上 ping 虚拟机，或者直接在计算节点的 qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539 名字空间中操作。如下所示。

```
   root@controller:~# ping -c 3 10.7.70.19
   root@controller:~# ssh cirros@10.7.70.19
      The authenticity of host '10.7.70.20 (10.7.70.20)' can't be
established.
      ECDSA key fingerprint is
SHA256:FcefDL2cTO3TTWAMkx5Ms82GdXRbCj/bO/mMyT6Ukho.
      Are you sure you want to continue connecting (yes/no)? yes
      Warning: Permanently added '10.7.70.20' (ECDSA) to the list of
known hosts.
      cirros@10.7.70.19's password:gocubsgo
      $

   # 或者:
   root@compute:~# ip netns exec
qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539 ping -c 3 10.7.70.19
      PING 10.7.70.19 (10.7.70.19) 56(84) bytes of data.
      64 bytes from 10.7.70.19: icmp_seq=1 ttl=64 time=14.9 ms
      64 bytes from 10.7.70.19: icmp_seq=2 ttl=64 time=1.49 ms
      64 bytes from 10.7.70.19: icmp_seq=3 ttl=64 time=0.474 ms
```

```
   # 从 qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539 名字空间登录虚拟机。
   root@compute:~# ip netns exec
qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539 ssh cirros@10.7.70.19
      cirros@10.7.70.19's password:gocubsgo
      $
```

通过 Web 控制台登录 OpenStack 虚拟机。

```
   root@controller:~# . metoo-openrc
   root@controller:~# openstack console url show provider-vm1
       +-------+-----------------------------------------------------
------------------------------+
       | Field | Value                                               |
       +-------+-----------------------------------------------------
------------------------------+
       | type  | novnc                                                |
       |              url                                              |
http://controller:6080/vnc_auto.html?token=889db4d8-5521-4f0d-8d10
-de8b3f7c65a1 |
       +-------+-----------------------------------------------------
------------------------------+
```

然后在其他计算机上用浏览器打开网址 http://controller:6080/vnc_auto.html? token=889db4d8-5521-4f0d-8d10-de8b3f7c65a1，即可进入虚拟机的控制台，输入用户 cirros 和密码 gocubsgo，即可登录 OpenStack 虚拟机。如果不能解析域名 controller，则无法打开这个网站，有两种方法可以解决这个问题：一是直接用 IP 地址 10.6.60.45 替换网址中的 controller，或者在 c:\windows\system32\ drivers\etc\hosts 中添加：10.6.60.45 controller。

（12）验证连通性。

由于虚拟机的网络不涉及控制节点，所以基于提供商网的虚拟机对外连通性如图 8-12 所示。

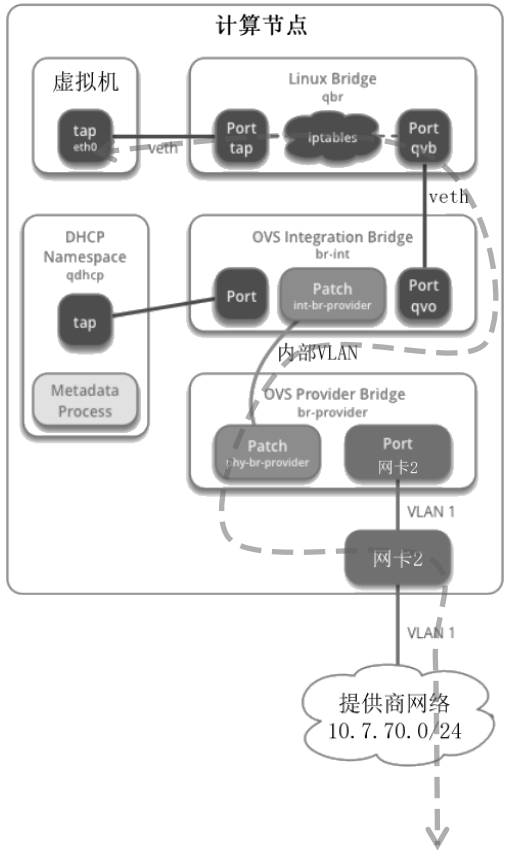


图 8-12　基于提供商网的虚拟机对外连通性

下面我们来看看计算节点中实际的虚拟网络是否符合图 8-10 的结构。

```
    # 检查是否存在一个 qdhcp 的网络名字空间：
    root@compute:~# ip netns list
        qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92

    # 进一步检查 qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92 网络名字空间
中的网卡配置：
    root@compute:~# ip netns exec
qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92 ifconfig
        tap1e683d9e-d6 Link encap:Ethernet  HWaddr fa:16:3e:6a:2a:d9
                inet     addr:10.7.70.11          Bcast:10.7.70.255
Mask:255.255.255.0
                inet6 addr: fe80::f816:3eff:fe6a:2ad9/64 Scope:Link

    # 检查是否存在一个 qbr 的 Linux 桥：
    root@compute:~# brctl show
```

```
        bridge name     bridge id           STP enabled     interfaces
        qbr53109655-7f  8000.2af631514da4     no       qvb53109655-7f
                                                        tap53109655-7f
        virbr0      8000.5254002d7bae       yes           virbr0-nic

    # 检查是否存在两个名为 br-int 和 br-provider 的 openvswitch 网桥（省略了部分输出）：
    root@compute:~# ovs-vsctl show
            Bridge br-provider
                Port "ens34"
                    Interface "ens34"
                Port phy-br-provider
                    Interface phy-br-provider
                        type: patch
                        options: {peer=int-br-provider}
            Bridge br-int
                Port "tap1e683d9e-d6"
                    tag: 1
                    Interface "tap1e683d9e-d6"
                        type: internal
                Port "qvo53109655-7f"
                    tag: 1
                    Interface "qvo53109655-7f"
                Port int-br-provider
                    Interface int-br-provider
                        type: patch
                        options: {peer=phy-br-provider}

    # 检查是否存在一个 OpenStack 虚拟机：
    root@compute:~# virsh list
        Id    Name                           State
        ----------------------------------------------------
         1    instance-00000001              running

    # 进一步检查 OpenStack 虚拟机的网卡：
    root@compute:~# virsh domiflist instance-00000001
        Interface     Type      Source        Model      MAC
        -------------------------------------------------------
        tap53109655-7f   bridge              qbr53109655-7f   virtio
fa:16:3e:56:0e:68
    # 采用 SSH 登录到 OpenStack 虚拟机中，然后运行命令 ifconfig 可以确认网卡的 MAC 地址就是 fa:16:3e:56:0e:68。
```

由此确定 Openstack 虚拟机与外网的通信路径为：

Openstack 虚拟机内网卡 eth0（MAC 地址：FA:16:3E:56:0E:68）←→

tap53109655-7f（MAC 地址：FA:16:3E:56:0E:68）←→ qvb53109655-7f←→ qvo53109655-7f←→ int-br-provider←→ phy-br-provider←→ens34←→外网。

下面在每个网元处抓包，进一步确认通信路径是否正确。

```
   root@controller:~# ssh cirros@10.7.70.20
       cirros@10.7.70.20's password:gocubsgo
       $ ping 10.7.70.45
       64 bytes from 10.7.70.45: seq=27 ttl=64 time=1.712 ms
       64 bytes from 10.7.70.45: seq=28 ttl=64 time=1.985 ms
       ……

   root@compute:~# tcpdump -ni ens34 -p icmp -v
       tcpdump: listening on ens34, link-type EN10MB (Ethernet),
capture size 262144 bytes
       21:29:24.917771 IP (tos 0x0, ttl 64, id 51775, offset 0, flags
[DF], proto ICMP (1), length 84)
          10.7.70.20 > 10.7.70.45: ICMP echo request, id 44545, seq
0, length 64
       21:29:24.918138 IP (tos 0x0, ttl 64, id 15268, offset 0, flags
[none], proto ICMP (1), length 84)
          10.7.70.45 > 10.7.70.20: ICMP echo reply, id 44545, seq 0,
length 64
       ……
   # 抓取一来一回的两个 ping 包。按“ctrl+c”中断 tcpdump 命令。

   root@compute:~# tcpdump -ni qvo53109655-7f -p icmp -v
      10.7.70.20 > 10.7.70.45: ICMP echo request, id 46337, seq 537,
length 64
       10.7.70.45 > 10.7.70.20: ICMP echo reply, id 46337, seq 537,
length 64

   root@compute:~# tcpdump -ni qvb53109655-7f -p icmp -v
       10.7.70.20 > 10.7.70.45: ICMP echo request, id 46337, seq 604,
length 64
       10.7.70.45 > 10.7.70.20: ICMP echo reply, id 46337, seq 604,
length 64

   root@compute:~# tcpdump -ni tap53109655-7f -p icmp -v
       10.7.70.20 > 10.7.70.45: ICMP echo request, id 46337, seq 737,
length 64
       10.7.70.45 > 10.7.70.20: ICMP echo reply, id 46337, seq 737,
length 64
```

2）self-service 网络

我们在 provider 网络的基础上做扩展，使其支持租户网络。为此需要增加一个网络节点，网络节点和计算节点各有三块网卡，分别接入管理网、提供商网和叠加网，如图 8-13 所示。

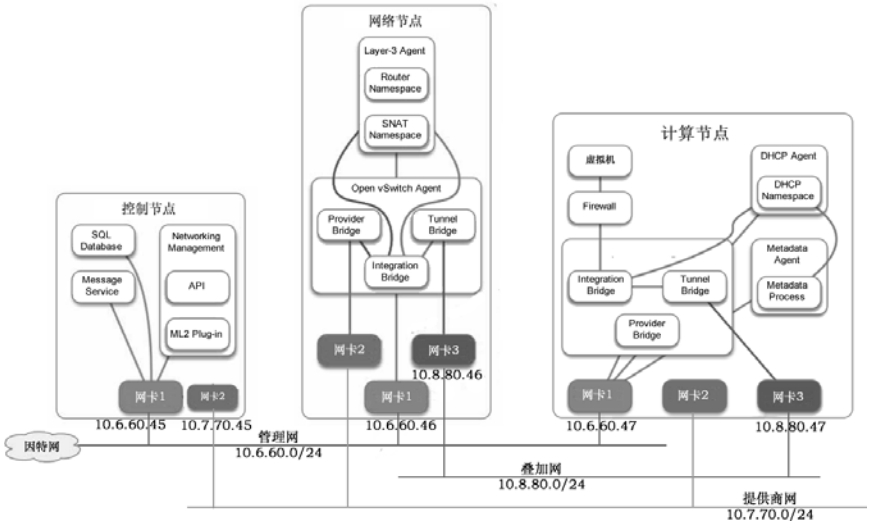


图 8-13　self-service 网络

由图 8-13 可知，计算节点在“Provider 网络”部分创建的提供商网桥（br-provider）没有用了，网卡 2 也没有加入该网桥，但是不能删除这个网桥，因为在后面启动 DVR 时要用到。OpenStack 虚拟机都是通过叠加网络到网络节点，然后由网络节点的提供商网桥的网卡 2 进出外网。

（1）在控制节点上修改一些参数。

```
root@controller:~# vim /etc/neutron/neutron.conf
  [DEFAULT]
  service_plugins = router
  allow_overlapping_ips = True

root@controller:~# vim /etc/neutron/plugins/ml2/ml2_conf.ini
  [ml2]
  type_drivers = flat,vlan,vxlan
  tenant_network_type = vxlan
  mechanism_drivers = openvswitch,l2population
```

```
    [ml2_type_vxlan]
    vni_ranges = 3:1000

  root@controller:~# systemctl restart neutron-server.service
```

（2）在网络节点上操作。

```
  root@network:~# apt -y install openvswitch-switch
neutron-plugin-ml2 neutron-l3-agent neutron-openvswitch-agent

  # 可把控制节点上的文件复制过来，然后确保如下两个参数。
  root@network:~# vim /etc/neutron/neutron.conf
    [DEFAULT]
    core_plugin = ml2
    auth_strategy = keystone

  root@network:~#                                          vim
/etc/neutron/plugins/ml2/openvswitch_agent.ini
    [ovs]
    bridge_mappings = my_physical_lan:br-provider
    local_ip = 10.8.80.46

    [agent]
    tunnel_types = vxlan
    l2_population = true

    [securitygroup]
    firewall_driver = iptables_hybrid

  root@network:~# vim /etc/neutron/l3_agent.ini
    [DEFAULT]
    interface_driver = openvswitch
    external_network_bridge =

  root@network:~# systemctl restart openvswitch-switch.service

  # 创建网桥br-provider。
  root@network:~# ovs-vsctl add-br br-provider
  root@network:~# ovs-vsctl add-port br-provider ens34

  root@network:~#  systemctl  restart  neutron-openvswitch-agent
neutron-l3-agent

  # 网络节点上的metadata服务没有用，所以禁用它。
  root@network:~# systemctl disable neutron-metadata-agent.service
```

（3）在计算节点上修改一些参数。

```
   # 计算节点上的 br-provider 暂时用不到，先断开其与网卡的连接，以免影响后面的
实验。
   root@controller:~# ovs-vsctl del-port br-provider ens34

   root@compute:~#                                                vim
/etc/neutron/plugins/ml2/openvswitch_agent.ini
     [ovs]
     local_ip = 10.8.80.47

     [agent]
     tunnel_types = vxlan
     l2_population = True

   root@compute:~# systemctl restart neutron-openvswitch-agent

   root@controller:~# . admin-openrc
   root@controller:~# openstack network agent list
       +-----+--------------------+---------+-------+-------+-----
--+---------------------------+
       | ID  | Agent Type         | Host    | Avail | Alive | State |
Binary                    |
       +-----+--------------------+---------+-------+-------+-----
--+---------------------------+
       | 2ac0| Open vSwitch agent | compute | None  | :-)   | UP    |
neutron-openvswitch-agent |
       | 6cf5| Metadata agent     | compute | None  | :-)   | UP    |
neutron-metadata-agent    |
       | 8c1a| DHCP agent         | compute | nova  | :-)   | UP    |
neutron-dhcp-agent        |
       | 8ecf| L3 agent           | network | nova  | :-)   | UP    |
neutron-l3-agent          |
       | ec2c| Open vSwitch agent | network | None  | :-)   | UP    |
neutron-openvswitch-agent |
       +-----+--------------------+---------+-------+-------+-----
--+---------------------------+
```

（4）创建网络、子网、路由器。

```
   root@controller:~# . admin-openrc

   # 把之前创建的提供商网络 provider-net1 改为外部网络。
   root@controller:~# openstack network set --external provider-net1

   root@controller:~# . metoo-openrc
```

```
   root@controller:~# openstack network create selfservice-net1
   root@controller:~# openstack subnet create --subnet-range
192.168.80.0/24 --network selfservice-net1 --dns-nameserver
202.96.128.166 selfservice-subnet1
   root@controller:~# openstack router create router1

   # 让路由器跨selfservice-subnet1和provider-net1两个网络。
   root@controller:~#   openstack   router   add   subnet   router1
selfservice-subnet1
   root@controller:~#  openstack  router  set  --external-gateway
provider-net1 router1

   # 查看路由器配置。
   root@controller:~# openstack router show router1

   root@network:~# ip netns list
     qrouter-5e7cc76e-0355-4240-9686-e4de9a5ac836
   root@compute:~# ip netns list
     qdhcp-57bddc7b-1e7e-4065-944b-7d77f6abe3d4
     qdhcp-e8d373b4-1308-43d3-8e01-fca5f41f2539
```

（5）创建并启动虚拟机。

```
   root@controller:~# . metoo-openrc
   root@controller:~# openstack network list
     +--------------------------------------+------------------+--
-----------------------------------+
     | ID                                   | Name             | Subnets   |
     +--------------------------------------+------------------+--
-----------------------------------+
     | 57bddc7b-1e7e-4065-944b-7d77f6abe3d4 | selfservice-net1 |
74f72b8c-2947-48b7-9ee0-c78ba61f7674 |
     | e8d373b4-1308-43d3-8e01-fca5f41f2539 | provider-net1    |
49407d5a-06e1-47ce-aede-745e9e6ae135 |
     +--------------------------------------+------------------+--
-----------------------------------+
   root@controller:~# openstack server create --flavor 1 --image
cirros --nic net-id=57bddc7b-1e7e-4065-944b-7d77f6abe3d4
selfservice-vm1
   root@controller:~# openstack server list
       +------+-----------------+--------+-----------------------
--------+--------+---------+
       | ID   | Name          | Status | Networks                  |
Image  | Flavor  |
       +------+-----------------+--------+-----------------------
--------+--------+---------+
```

```
       |     2e33     |     selfservice-vm1     |     ACTIVE     |
selfservice-net1=192.168.80.9  | cirros | m1.nano |
       | 9ea4 | provider-vm1    | ACTIVE | provider-net1=10.7.70.19
| cirros | m1.nano |
       +------+-----------------+--------+----------------------
--------+--------+---------+
```

发现刚刚创建的虚拟机被分配 IP 地址 192.168.80.9，这个地址在 selfservice-net1 的一个子网上，这是一个内部 IP 地址，显然外面无法直接访问这个 IP 地址，但是从虚拟机里面可以连通外网，因为网络节点会做 SNAT 转换。为了能让外面访问进来，我们需要给虚拟机分配一个提供商网络上的 IP 地址（称为浮动 IP 地址）。

（6）给虚拟机分配一个浮动 IP 地址。

```
   root@controller:~# . metoo-openrc

   # 创建浮动 IP 地址。
   root@controller:~# openstack floating ip create provider-net1
      +---------------------+--------------------------------------
-+
      | Field             | Value                              |
      +---------------------+--------------------------------------
-+
      | fixed_ip_address    | None                               |
      | floating_ip_address | 10.7.70.18                           |
      | floating_network_id | e8d373b4-1308-43d3-8e01-fca5f41f2539 |
      | id                  | 4f1c3f92-f825-4a1e-aa94-ad88fa1b6245 |
      | name                | 10.7.70.18                         |
      +---------------------+--------------------------------------
-+

   # 把浮动 IP 分配给虚拟机。
   root@controller:~#   openstack   server   add   floating   ip
selfservice-vm1 10.7.70.18
   root@controller:~# openstack server list
       +------+-----------------+--------+----------------------
--------+--------+---------+
       | ID   | Name          | Status | Networks                   |
Image  | Flavor  |
       +------+-----------------+--------+----------------------
--------+--------+---------+
       |     2e33     |     selfservice-vm1     |     ACTIVE     |
selfservice-net1=192.168.80.9, | cirros | m1.nano |
```

```
       |     |              |       |  10.7.70.18                |
|        |
       | 9ea4 | provider-vm1    | SHUTOFF| provider-net1=10.7.70.19
| cirros | m1.nano |
       +------+-----------------+--------+-----------------------
--------+--------+---------+
```

（7）登录 OpenStack 虚拟机。

```
   root@controller:~# ping -c 2 10.7.70.18
      PING 10.7.70.14 (10.7.70.18) 56(84) bytes of data.
      64 bytes from 10.7.70.18: icmp_seq=1 ttl=63 time=12.8 ms
      64 bytes from 10.7.70.18: icmp_seq=2 ttl=63 time=1.30 ms

   root@controller:~# ssh cirros@10.7.70.18
      The authenticity of host '10.7.70.18 (10.7.70.18)' can't be
established.
      ECDSA             key             fingerprint             is
SHA256:By3C4cUooV/+/VPieIyh6vRoD7xlLX9I6IUYvkhdv20.
      Are you sure you want to continue connecting (yes/no)? yes
      Warning: Permanently added '10.7.70.18' (ECDSA) to the list of
known hosts.
      cirros@10.7.70.14's password:gocubsgo
      $ ifconfig
      eth0     Link encap:Ethernet  HWaddr FA:16:3E:66:A5:C3
               inet    addr:192.168.80.8       Bcast:192.168.80.255
Mask:255.255.255.0
               inet6 addr: fe80::f816:3eff:fe66:a5c3/64 Scope:Link
```

发现 OpenStack 虚拟机里面的网卡 IP 地址在内部局域网 192.168.80. 0/24 中，此局域网通过路由器连接提供商网络。由路由器完成 SNAT 转换，即 10.7.70.18←→192.168.80.8。

（8）验证连通性。

OpenStack 虚拟机与外网的连通性如图 8-14 所示。

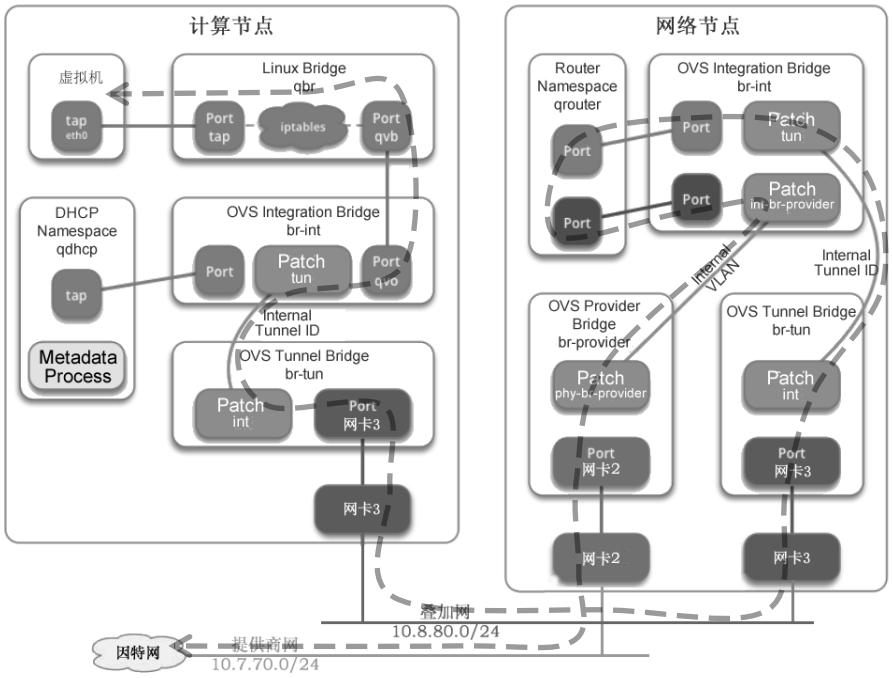


图 8-14　OpenStack 虚拟机与外网的连通性

首先查验网络节点上的各个网元及其之间的连通性。

```
   # 是否存在路由器名字空间：
   root@network:~# ip netns list
      qrouter-a38f9e5e-fe1c-4f9b-906e-bbb9a5fc87c2

   # 进一步查看路由器名字空间中的网卡：
   root@network:~#            ip            netns            exec
qrouter-a38f9e5e-fe1c-4f9b-906e-bbb9a5fc87c2 ifconfig
      lo       Link encap:Local Loopback
               inet addr:127.0.0.1  Mask:255.0.0.0
               …
      qg-16048f46-2c Link encap:Ethernet  HWaddr fa:16:3e:c1:42:7b
               inet      addr:10.7.70.19          Bcast:10.7.70.255
Mask:255.255.255.0
               inet6 addr: fe80::f816:3eff:fec1:427b/64 Scope:Link
               …
      qr-369eaae6-4f Link encap:Ethernet  HWaddr fa:16:3e:6a:7b:86
               inet    addr:192.168.80.1       Bcast:192.168.80.255
Mask:255.255.255.0
               inet6 addr: fe80::f816:3eff:fe6a:7b86/64 Scope:Link
               …
```

```
   # 是否存在三个 OVS 网桥：
   root@network:~# ovs-vsctl show
      Bridge br-tun
         Port patch-int
            Interface patch-int
               type: patch
               options: {peer=patch-tun}
         Port "vxlan-0a08502f"
            Interface "vxlan-0a08502f"
               type: vxlan
               options:     {df_default="true",      in_key=flow,
local_ip="10.8.80.46", out_key=flow, remote_ip="10.8.80.47"}
      Bridge br-int
         Port int-br-provider
            Interface int-br-provider
               type: patch
               options: {peer=phy-br-provider}
         Port "qg-16048f46-2c"
            tag: 2
            Interface "qg-16048f46-2c"
               type: internal
         Port "qr-369eaae6-4f"
            tag: 1
            Interface "qr-369eaae6-4f"
               type: internal
         Port patch-tun
            Interface patch-tun
               type: patch
               options: {peer=patch-int}
      Bridge br-provider
         Port phy-br-provider
            Interface phy-br-provider
               type: patch
               options: {peer=int-br-provider}
         Port "ens34"
            Interface "ens34"
```

由此可见，网络节点内部各个虚拟网元及它们之间的连通性正常，下面再来看看计算节点内部的虚拟网络情况。

```
   # 是否存在两个 DHCP 名字空间：
   root@compute:~# ip netns list
      qdhcp-b3c314b2-8bcf-4c92-bcc4-6bf93949190a
      qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92

   # 进一步查看各个名字空间中的网卡：
   root@compute:~#            ip            netns            exec
```

```
qdhcp-b3c314b2-8bcf-4c92-bcc4-6bf93949190a ifconfig
      lo       Link encap:Local Loopback
               inet addr:127.0.0.1  Mask:255.0.0.0
               …
      tap697450f6-d5 Link encap:Ethernet  HWaddr fa:16:3e:e1:56:26
               inet    addr:192.168.80.2       Bcast:192.168.80.255
Mask:255.255.255.0
               inet6 addr: fe80::f816:3eff:fee1:5626/64 Scope:Link
               …
   root@compute:~# ip netns exec
qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92 ifconfig
      lo       Link encap:Local Loopback
               inet addr:127.0.0.1  Mask:255.0.0.0
               …
      tap1e683d9e-d6 Link encap:Ethernet  HWaddr fa:16:3e:6a:2a:d9
               inet     addr:10.7.70.11          Bcast:10.7.70.255
Mask:255.255.255.0
               inet6 addr: fe80::f816:3eff:fe6a:2ad9/64 Scope:Link
               …

   # 是否存在三个 Linux 网桥（其中第一个网桥就是虚拟机 selfservice-vm1 用到
的）：
   root@compute:~# brctl show
      bridge name     bridge id             STP enabled     interfaces
      qbr3d856251-75      8000.0af41c20b113                     no
qvb3d856251-75

tap3d856251-75
      qbr53109655-7f      8000.3ef6027f1db9                     no
qvb53109655-7f
      virbr0                    8000.5254002d7bae             yes
virbr0-nic

   # 查看正在运行的 OpenStack 虚拟机：
   root@compute:~# virsh list
      Id    Name                         State
      ----------------------------------------------------
       1    instance-00000002            running

   # 进一步检查 OpenStack 虚拟机的网卡：
   root@compute:~# virsh domiflist instance-00000002
      Interface     Type      Source        Model      MAC
      -------------------------------------------------------------
----------
      tap3d856251-75   bridge              qbr3d856251-75   virtio
fa:16:3e:66:a5:c3

   # 检查 OVS 网桥（其中 br-provider 没用到，显示结果中被略去）：
```

```
   root@compute:~# ovs-vsctl show
      Bridge br-tun
         Port "vxlan-0a08502e"
            Interface "vxlan-0a08502e"
               type: vxlan
               options:     {df_default="true",     in_key=flow,
local_ip="10.8.80.47", out_key=flow, remote_ip="10.8.80.46"}
         Port patch-int
            Interface patch-int
               type: patch
               options: {peer=patch-tun}
      Bridge br-int
         Port "qvo3d856251-75"
            tag: 2
            Interface "qvo3d856251-75"
         Port "tap1e683d9e-d6"
            tag: 1
            Interface "tap1e683d9e-d6"
               type: internal
         Port "qvo53109655-7f"
            tag: 1
            Interface "qvo53109655-7f"
         Port "tap697450f6-d5"
            tag: 2
            Interface "tap697450f6-d5"
               type: internal
         Port patch-tun
            Interface patch-tun
               type: patch
               options: {peer=patch-int}
```

由此可以确定 Openstack 虚拟机与外网的通信路径如下：

Openstack 虚拟机内网卡 eth0←→tap3d856251-75←→qvb3d856251-75←→qvo3d856251-75←→patch-tun←→patch-int←→vxlan-0a08502e ←→ 计算节点上的网卡 3←→ 网络节点上的网卡 3←→vxlan-0a08502f ←→patch-int←→patch-tun←→qr-369eaae6-4f←→qg-16048f46-2c←→ int-br- provider ←→phy-br-provider←→网络节点上的网卡 2←→提供商网络。

其中下画线部分是在计算节点上，无下画线部分在网络节点上。下面分别在路径的几个关键点抓包来进一步确认通信路径是否正确。

```
   # SSH 登录到 OpenStack 虚拟机并跟踪路由。
```

```
   root@controller:~# ssh cirros@10.7.70.18
   cirros@10.7.70.18's password:gocubsgo
   $ traceroute 10.7.70.45
      1  192.168.80.1 (192.168.80.1)  3.939 ms  15.163 ms  12.240 ms
      2  10.7.70.45 (10.7.70.45)  31.526 ms  7.218 ms  4.668 ms

   # 在控制节点上不断 ping 虚拟机。
   root@controller:~# ping 10.7.70.18
       64 bytes from 10.7.70.18: icmp_seq=1947 ttl=63 time=1.63 ms
       64 bytes from 10.7.70.18: icmp_seq=1948 ttl=63 time=1.80 ms
       …

   # 在计算节点上抓包。
   root@compute:~# tcpdump -ni ens35 -v          ←叠加网上的 vxlan 包
       tcpdump: listening on ens35, link-type EN10MB (Ethernet),
capture size 262144 bytes
       17:40:37.734489 IP (tos 0x0, ttl 64, id 11105, offset 0, flags
[DF], proto UDP (17), length 134)
           10.8.80.46.43645 > 10.8.80.47.4789: VXLAN, flags [I] (0x08),
vni 14
       IP (tos 0x0, ttl 63, id 61145, offset 0, flags [DF], proto ICMP
(1), length 84)
           10.7.70.45 > 192.168.80.8: ICMP echo request, id 4492, seq
1829, length 64

   root@compute:~# tcpdump -ni qvo3d856251-75 -p icmp -v
       10.7.70.45 > 192.168.80.8: ICMP echo request, id 4492, seq 2254,
length 64
       192.168.80.8 > 10.7.70.45: ICMP echo reply, id 4492, seq 2254,
length 64

   root@compute:~# tcpdump -ni qvb3d856251-75 -p icmp -v
       10.7.70.45 > 192.168.80.8: ICMP echo request, id 4492, seq 2334,
length 64
       192.168.80.8 > 10.7.70.45: ICMP echo reply, id 4492, seq 2334,
length 64

   root@compute:~# tcpdump -ni tap3d856251-75 -p icmp -v
       10.7.70.45 > 192.168.80.8: ICMP echo request, id 4492, seq 2460,
length 64
       192.168.80.8 > 10.7.70.45: ICMP echo reply, id 4492, seq 2460,
length 64

   # 在网络节点上抓包（需要等好几秒钟才有反应）。
   root@network:~# tcpdump -ni ens35 -v          ←叠加网上的 vxlan 包
       10.8.80.46.43645 > 10.8.80.47.4789: VXLAN, flags [I] (0x08),
vni 14
       10.7.70.45 > 192.168.80.8: ICMP echo request, id 4492, seq 2582,
```

```
length 64

   # 抓取路由器中端口 qr-369eaae6-4f 的包。
   root@network:~#            ip            netns            exec
qrouter-a38f9e5e-fe1c-4f9b-906e-bbb9a5fc87c2      tcpdump      -ni
qr-369eaae6-4f -p icmp -v
       10.7.70.45 > 192.168.80.8: ICMP echo request, id 42241, seq 169,
length 64
       192.168.80.8 > 10.7.70.45: ICMP echo reply, id 42241, seq 169,
length 64

   root@compute:~# tcpdump -ni ens34 -p icmp -v
       10.7.70.45 > 10.7.70.18: ICMP echo request, id 4492, seq 2657,
length 64
       10.7.70.14 > 10.7.70.45: ICMP echo reply, id 4492, seq 2657,
length 64
```

通过在通信路径的关键点抓包，验证路径正确，而且进一步验证了网络节点上的虚拟路由器做了 NAT。

3）启用 DVR 功能。

采用 self-service 网络，计算节点上的外出网络流量要走网络节点，这样网络节点很可能成为性能的瓶颈，而且是单点故障。现在的 OpenStack 版本支持计算节点的南北流量直接走提供商网络进出，而不需要或很少需要网络节点参与。这样有两大好处：一是降低了网络节点的负担且剔除了其单点故障特性，二是每个计算节点的网络流量直接出去从而增大了整个云端的吞吐量。接下来我们在 self-service 网络的基础上启用 DVR 功能，尽管开启了 DVR，但是分配了固定 IP 地址的虚拟机，其南北流量还是走网络节点。带 DVR 的三节点部署如图 8-5 所示。

（1）在控制节点上修改一些参数。

```
   root@controller:~# vim /etc/neutron/neutron.conf
     [DEFAULT]
     router_distributed = True

   root@controller:~# systemctl restart neutron-server
```

（2）在网络节点上修改一些参数。

```
   root@network:~#                                                vim
```

```
/etc/neutron/plugins/ml2/openvswitch_agent.ini
     [DEFAULT]
     enable_distributed_routing = True

   root@network:~# vim /etc/neutron/l3_agent.ini
     [DEFAULT]
     agent_mode = dvr_snat

   root@controller:~# systemctl restart neutron-openvswitch-agent
l3-agent
```

（3）在计算节点上修改一些参数。

```
   root@compute:~#                                                vim
/etc/neutron/plugins/ml2/openvswitch_agent.ini
     [DEFAULT]
     enable_distributed_routing = True

   root@compute:~# apt-get install neutron-l3-agent -y
   root@compute:~# vim /etc/neutron/l3_agent.ini
     [DEFAULT]
     interface_driver = openvswitch
     external_network_bridge =
     agent_mode = dvr

   root@compute:~# ovs-vsctl add-port br-provider ens34
   root@compute:~#  systemctl  restart  neutron-openvswitch-agent
neutron-l3-agent
```

（4）查看网络服务运行状态。

```
   root@controller:~# . admin-openrc
   root@controller:~# openstack network agent list
       +------+--------------------+---------+-------+-------+----
---+---------------------------+
       | ID   | Agent Type         | Host    | Avail | Alive | State |
Binary                    |
       +------+--------------------+---------+-------+-------+----
---+---------------------------+
       | 11d4f| DHCP agent         | compute | nova  | :-)   | UP    |
neutron-dhcp-agent        |
       | 6d2bf| L3 agent           | compute | nova  | :-)   | UP    |
neutron-l3-agent          |
       | 93ec7| Metadata agent     | compute | None  | :-)   | UP    |
neutron-metadata-agent    |
       | ca9e7| Open vSwitch agent | compute | None  | :-)   | UP    |
neutron-openvswitch-agent |
       | d5a0e| Open vSwitch agent | network | None  | :-)   | UP    |
neutron-openvswitch-agent |
```

```
       | e8283| L3 agent         | network | nova  | :-)   | UP    |
neutron-l3-agent        |
       +------+------------------+---------+-------+-------+----
---+---------------------------+
```

（5）创建网络、子网、路由器。

```
   root@controller:~# . metoo-openrc

   root@controller:~# openstack network create selfservice-net2
   root@controller:~# openstack subnet create --subnet-range
192.168.89.0/24 --network selfservice-net2 --dns-nameserver 8.8.8.8
selfservice-subnet2
   root@controller:~# openstack router create router2

   # 让路由器跨selfservice-subnet2和provider-net1两个网络。
   root@controller:~#   openstack   router   add   subnet   router2
selfservice-subnet2
   root@controller:~# openstack router set router2 --external-gateway
provider-net1

   # 查看路由器配置。
   root@controller:~# . admin-openrc
   root@controller:~# openstack router show router2
      +-------------------------+---------+
      | Field                   | Value   |
      +-------------------------+---------+
      | admin_state_up          | UP      |
      | distributed             | True    |
      | ha                      | False   |
      | name                    | router2 |
      | revision_number         | 1       |
      | status                  | ACTIVE  |
      +-------------------------+---------+

   root@network:~# ip netns list
     snat-d5f2f382-1ae6-4575-81b1-381ba1f9902b
     qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b
     qrouter-a38f9e5e-fe1c-4f9b-906e-bbb9a5fc87c2

   root@compute:~# ip netns list
     fip-7a550a04-e157-4b84-a90d-8753aed05a92
     qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b
     qdhcp-83b1fe24-f0e5-474d-a566-f9b5db2a31d8
     qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92
     qdhcp-b3c314b2-8bcf-4c92-bcc4-6bf93949190a
```

注意计算节点和网络节点上存在同名路由器名字空间，说明这个路由器

是跨越两台机器的。

（6）创建并启动虚拟机。

```
   root@controller:~# . metoo-openrc
   root@controller:~# openstack network list
     +--------------------------------------+------------------+--
---------------+
     | ID                                   | Name           | Subnets  |
     +--------------------------------------+------------------+--
---------------+
     |  7a550a04-e157-4b84-a90d-8753aed05a92  |  provider-net1     |
8b02ee36-7c1b-6 |
     |  83b1fe24-f0e5-474d-a566-f9b5db2a31d8  |  selfservice-net2  |
76cdd83f-4ff0-9 |
     |  b3c314b2-8bcf-4c92-bcc4-6bf93949190a  |  selfservice-net1  |
0e45a8bf-fe72-c |
     +--------------------------------------+------------------+--
---------------+

   root@controller:~# openstack server create --flavor 1 --image
cirros --nic net-id=83b1fe24-f0e5-474d-a566-f9b5db2a31d8
selfservice-vm3
       +-----+-----------------+---------+-----------------------
--------+--------+---------+
       | ID  | Name          | Status  | Networks                  |
Image  | Flavor  |
       +-----+-----------------+---------+-----------------------
--------+--------+---------+
       | 82d8| selfservice-vm3 | ACTIVE  |
selfservice-net2=192.168.89.8  | cirros | m1.nano |
       | 8c4a| selfservice-vm1 | SHUTOFF |
selfservice-net1=192.168.80.8, | cirros | m1.nano |
       |    |              |        |  10.7.70.14               |
|        |
       | 80d4| provider-vm1    | SHUTOFF | provider-net1=10.7.70.20
| cirros | m1.nano |
       +-----+-----------------+---------+-----------------------
--------+--------+---------+

   root@controller:~# openstack floating ip create provider-net1
      +---------------------+-------------------------------------
-+
      | Field             | Value                               |
      +---------------------+-------------------------------------
-+
      | floating_ip_address | 10.7.70.16                          |
      | floating_network_id | 7a550a04-e157-4b84-a90d-8753aed05a92 |
```

```
      | name                 | 10.7.70.16                           |
      +----------------------+--------------------------------------
-+

   root@controller:~# openstack server add floating ip
selfservice-vm3 10.7.70.16
```

现在可以 ping 通 10.7.70.16，或者 SSH 登录这台虚拟机了。

（7）验证连通性。

启用 DVR 的 OpenStack 虚拟机与外网的连通性如图 8-15 所示。

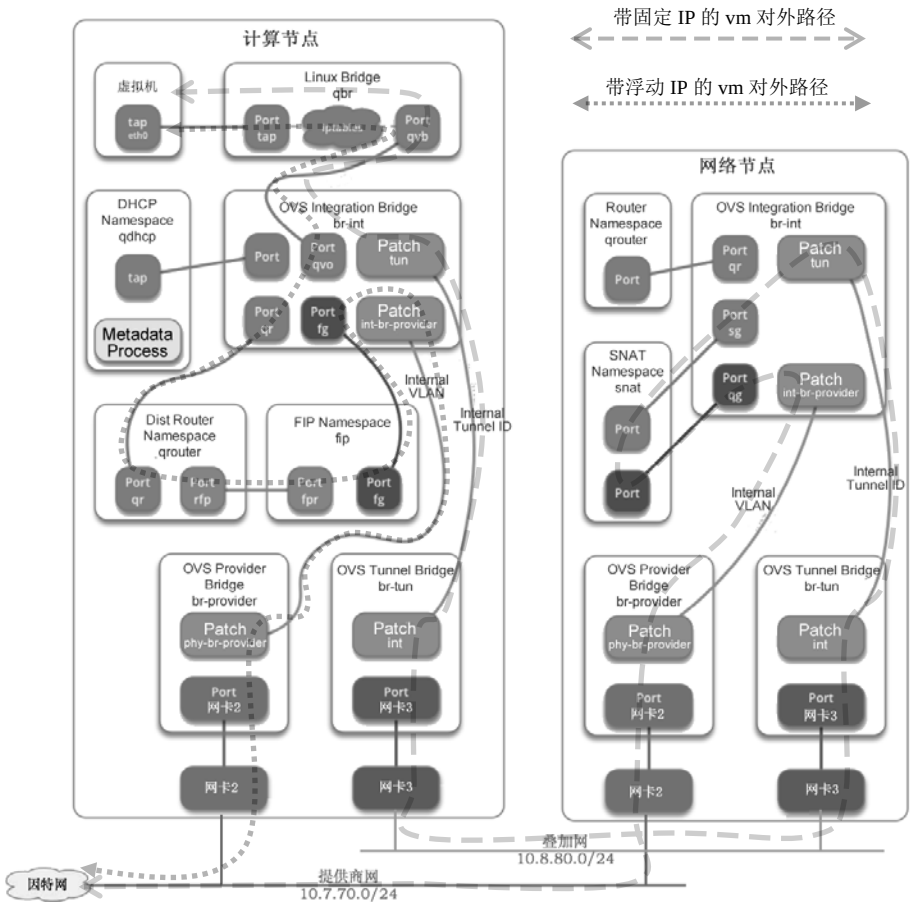


图 8-15　启用 DVR 的 OpenStack 虚拟机与外网的连通性

下面检查各个节点中实际的虚拟网元状况，先看计算节点。

```
   # 至少存在三个网络名字空间（分别是浮动 IP 地址、路由和 DHCP）：
   root@compute:~# ip netns list
       fip-7a550a04-e157-4b84-a90d-8753aed05a92
       qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b
       qdhcp-b3c314b2-8bcf-4c92-bcc4-6bf93949190a
       qdhcp-83b1fe24-f0e5-474d-a566-f9b5db2a31d8
       qdhcp-7a550a04-e157-4b84-a90d-8753aed05a92

   # 进一步查看浮动 IP 地址、路由和 DHCP 网络名字空间中的网元：
   root@compute:~#            ip            netns            exec
fip-7a550a04-e157-4b84-a90d-8753aed05a92 ifconfig
       fg-41c08d72-fc Link encap:Ethernet  HWaddr fa:16:3e:96:89:d9
                inet     addr:10.7.70.17          Bcast:10.7.70.255
Mask:255.255.255.0
                …
       fpr-d5f2f382-1 Link encap:Ethernet  HWaddr ba:7d:b0:a6:35:62
                inet     addr:169.254.106.115         Bcast:0.0.0.0
Mask:255.255.255.254
                …
   root@compute:~# ip netns exec
qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b ifconfig
       qr-0d5b92c1-ad Link encap:Ethernet  HWaddr fa:16:3e:99:62:11
                inet    addr:192.168.89.1       Bcast:192.168.89.255
Mask:255.255.255.0
                …
       rfp-d5f2f382-1 Link encap:Ethernet  HWaddr 16:7f:22:67:c3:87
                inet     addr:169.254.106.114         Bcast:0.0.0.0
Mask:255.255.255.254
                …
   root@compute:~# ip netns exec
qdhcp-83b1fe24-f0e5-474d-a566-f9b5db2a31d8 ifconfig
       tap525c45f1-14 Link encap:Ethernet  HWaddr fa:16:3e:bc:ee:a2
                inet    addr:192.168.89.2       Bcast:192.168.89.255
Mask:255.255.255.0
                …

   # 是否存在一个 Linux 网桥：
   root@compute:~# brctl show
       bridge  name       bridge  id                     STP  enabled
interfaces
       …
       qbre0309ead-87      8000.c255c21308fd                    no
qvbe0309ead-87

tape0309ead-87
       …

   # 检查虚拟机的网卡：
```

```
   root@compute:~# virsh list
      Id    Name                        State
      ----------------------------------------------
      1     instance-00000003             running

   root@compute:~# virsh domiflist instance-00000003
      Interface     Type      Source        Model      MAC
      -------------------------------------------------------------
----------
      tape0309ead-87   bridge              qbre0309ead-87   virtio
fa:16:3e:e7:d0:f6

   # 检查是否存在 br-tun、br-int、br-provider 三个 OVS 网桥（省略了无关内容）：
   root@compute:~# ovs-vsctl show
       Bridge br-tun
          Port "vxlan-0a08502e"
             Interface "vxlan-0a08502e"
                type: vxlan
                options:     {df_default="true",     in_key=flow,
local_ip="10.8.80.47", out_key=flow, remote_ip="10.8.80.46"}
          Port patch-int
             Interface patch-int
                type: patch
                options: {peer=patch-tun}
       Bridge br-provider
          Port phy-br-provider
             Interface phy-br-provider
                type: patch
                options: {peer=int-br-provider}
          Port "ens34"
             Interface "ens34"
       Bridge br-int
          Port "tap525c45f1-14"
             tag: 2
             Interface "tap525c45f1-14"
                type: internal
          Port "qvoe0309ead-87"
             tag: 2
             Interface "qvoe0309ead-87"
          Port "qr-0d5b92c1-ad"
             tag: 2
             Interface "qr-0d5b92c1-ad"
                type: internal
          Port "fg-41c08d72-fc"
             tag: 1
             Interface "fg-41c08d72-fc"
                type: internal
          Port int-br-provider
```

```
            Interface int-br-provider
                type: patch
                options: {peer=phy-br-provider}
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
```

接下来检查网络节点。

```
    # 至少存在两个网络名字空间（分别是 SNAT 和路由）：
    root@network:~# ip netns list
        snat-d5f2f382-1ae6-4575-81b1-381ba1f9902b
        qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b
        qrouter-a38f9e5e-fe1c-4f9b-906e-bbb9a5fc87c2

    root@network:~#            ip            netns            exec
snat-d5f2f382-1ae6-4575-81b1-381ba1f9902b ifconfig
        qg-4d570a87-bb Link encap:Ethernet  HWaddr fa:16:3e:e6:3e:62
                inet     addr:10.7.70.15          Bcast:10.7.70.255
Mask:255.255.255.0
                …
        sg-cfa2ff2a-63 Link encap:Ethernet  HWaddr fa:16:3e:4f:e9:84
                inet    addr:192.168.89.5       Bcast:192.168.89.255
Mask:255.255.255.0
                …
    root@network:~# ip netns exec
qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b ifconfig
        qr-0d5b92c1-ad Link encap:Ethernet  HWaddr fa:16:3e:99:62:11
                inet    addr:192.168.89.1       Bcast:192.168.89.255
Mask:255.255.255.0
                …

    # 检查是否存在 br-tun、br-int、br-provider 三个 OVS 网桥（省略了无关内容）：
    root@compute:~# ovs-vsctl show
        Bridge br-tun
            Port patch-int
                Interface patch-int
                    type: patch
                    options: {peer=patch-tun}
            Port "vxlan-0a08502f"
                Interface "vxlan-0a08502f"
                    type: vxlan
                    options:     {df_default="true",     in_key=flow,
local_ip="10.8.80.46", out_key=flow, remote_ip="10.8.80.47"}
        Bridge br-int
            Port "qr-0d5b92c1-ad"
                tag: 2
```

```
            Interface "qr-0d5b92c1-ad"
                type: internal
        Port "sg-cfa2ff2a-63"
            tag: 2
            Interface "sg-cfa2ff2a-63"
                type: internal
        Port int-br-provider
            Interface int-br-provider
                type: patch
                options: {peer=phy-br-provider}
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port "qg-4d570a87-bb"
            tag: 1
            Interface "qg-4d570a87-bb"
                type: internal
    Bridge br-provider
        Port phy-br-provider
            Interface phy-br-provider
                type: patch
                options: {peer=int-br-provider}
        Port "ens34"
            Interface "ens34"
```

在启用 DVR 的 SelfService 网络中，被分配了固定 IP 地址的 OpenStack 虚拟机和被分配了浮动 IP 地址的虚拟机，它们与外网通信的路径是不同的，固定 IP 地址的 OpenStack 虚拟机继续绕到网络节点出去，而浮动 IP 地址的虚拟机直接走计算节点的网卡（此网卡连接提供商网络）出去。因此浮动 IP 地址的虚拟机的通信路径如下。

OpenStack 虚拟机内网卡 eth0←→ tape0309ead-87←→ qvbe0309ead-87←→qvoe0309ead-87←→qr-0d5b92c1-ad←→rfp-d5f2f382-1←→fpr-d5f2f382-1←→fg-41c08d72-fc←→ int-br-provider←→ phy-br-provide r←→计算节点上的网卡 2←→提供商网络。

固定 IP 的虚拟机的通信路径如下。

Openstack 虚拟机内网卡 eth0←→ tape0309ead-87←→ qvbe0309ead-87←→qvoe0309ead-87←→int-br-provider←→patch-tun←→patch-int←→vxlan-

0a08502e←→ 计算节点上的网卡 3←→网络节点上的网卡 3←→vxlan-0a08502f←→patch-int←→patch-tun←→sg-cfa2ff2a-63←→qg-4d570a87-bb←→int-br-provider←→phy-br-provider←→网络节点上的网卡 2←→提供商网络。

下面分别在路径的几个关键点抓包来进一步确认通信路径是否正确。这次只验证浮动 IP 地址。

```
   # SSH 登录 OpenStack 虚拟机并跟踪路由。
   root@controller:~# ssh cirros@10.7.70.18
   cirros@10.7.70.18's password:gocubsgo
   $ traceroute 10.7.70.45
      traceroute to 10.7.70.45 (10.7.70.45), 30 hops max, 46 byte
packets
       1  192.168.89.1 (192.168.89.1)  2.488 ms  24.478 ms  0.569 ms
       2   169.254.106.115 (169.254.106.115)   2.332  ms   6.880  ms
0.615 ms
       3  10.7.70.45 (10.7.70.45)  14.598 ms  31.450 ms  2.051 ms

   # 不断发出 ping 包，以便在关键点抓包。
   $ ping 10.7.70.45
       64 bytes from 10.7.70.45: seq=0 ttl=62 time=7.830 ms
       64 bytes from 10.7.70.45: seq=1 ttl=62 time=3.922 ms
       …

   # 在计算节点上抓包。
   root@compute:~# tcpdump -ni tape0309ead-87 -p icmp -v
      192.168.89.8 > 10.7.70.45: ICMP echo request, id 42241, seq 28,
length 64
      10.7.70.45 > 192.168.89.8: ICMP echo reply, id 42241, seq 28,
length 64
   root@compute:~# tcpdump -ni qvbe0309ead-87 -p icmp -v
       192.168.89.8 > 10.7.70.45: ICMP echo request, id 42241, seq 159,
length 64
       10.7.70.45 > 192.168.89.8: ICMP echo reply, id 42241, seq 159,
length 64

   root@compute:~# tcpdump -ni qvoe0309ead-87 -p icmp -v
       192.168.89.8 > 10.7.70.45: ICMP echo request, id 42241, seq 297,
length 64
       10.7.70.45 > 192.168.89.8: ICMP echo reply, id 42241, seq 297,
length 64

   root@compute:~# tcpdump -ni ens34 -p icmp -v
       10.7.70.18 > 10.7.70.45: ICMP echo request, id 42241, seq 407,
```

```
length 64
       10.7.70.45 > 10.7.70.18: ICMP echo reply, id 42241, seq 407,
length 64

   # 在控制节点上抓包。
   root@controller:~# tcpdump -ni ens34 -p icmp -v
       10.7.70.18 > 10.7.70.45: ICMP echo request, id 42241, seq 510,
length 64
       10.7.70.45 > 10.7.70.18: ICMP echo reply, id 42241, seq 510,
length 64
```

这样就验证了浮动 IP 地址的虚拟机与外网的通信路径是正确的。可以进一步在网络节点上抓包或者在计算节点上的叠加网卡抓取 VXLAN 包，如果能抓到包，说明有问题。

至于验证固定 IP 地址的虚拟机出外网的路径问题，首先要给虚拟机添加网卡，然后进入 OpenStack 虚拟机里配置网卡 IP。然后要拆卸原有的网卡。大致的命令如下（注意：固定的 IP 地址是属于提供商网络的）。

```
   root@controller:~# . metoo-openrc
   root@controller:~# openstack server add fixed ip
--fixed-ip-address 10.7.70.18 selfservice-vm3 provider-net1
   root@controller:~# nova interface-list selfservice-vm3
   +------------+--------------------------------------+--------+-
--------------+-------------------+
   | Port State | Port ID                              | Net ID | IP
addresses  | MAC Addr         |
   +------------+--------------------------------------+--------+-
--------------+-------------------+
   | ACTIVE     | babd903e-7a35-4958-8da9-9f6ff6300668 | 7a550a |
10.7.70.18    | fa:16:3e:59:33:c4 |
   | ACTIVE     | c80c1e03-a749-430f-a183-766673515b67 | 83b1fe |
192.168.89.10 | fa:16:3e:72:a3:1e |
   +------------+--------------------------------------+--------+-
--------------+-------------------+
   root@compute:~#            ip            netns            exec
qrouter-d5f2f382-1ae6-4575-81b1-381ba1f9902b                    ssh
cirros@192.168.89.10
   cirros@ 192.168.89.10's password:gocubsgo
   $ sudo ifconfig eth1 add 10.7.70.18 netmask 255.255.255.0 up
```

此时在控制节点上应该能 ping 通 10.7.70.18 了，接下来要把原来的浮动 IP 地址删掉或者干脆把网卡拆卸下来，采用如下命令完成。

```
   root@controller:~# openstack server remove floating ip
selfservice-vm3 10.7.70.16

   # 或者:
   root@controller:~# nova interface-detach selfservice-vm3
c80c1e03-a749-430f-a183-766673515b67
```

4）安装 Horizon

以下操作都在控制节点上进行。

```
   root@controller:~# apt install OpenStack-dashboard

   # 修改一些参数的值。
   root@controler:~# vim /etc/OpenStack-dashboard/local_settings.py

     OPENSTACK_HOST = "controller"

            'LOCATION': 'controller:11211',

     OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True

     OPENSTACK_API_VERSIONS = {
        "identity": 3,
        "image": 2,
        "volume": 2,
        "compute": 2,
     }

     OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = 'Default'

     OPENSTACK_NEUTRON_NETWORK = {
        'enable_router': True,
        'enable_quotas': True,
        'enable_ipv6': False,
        'enable_distributed_router': True,
        'enable_ha_router': False,
        'enable_fip_topology_check': True,

     TIME_ZONE = "Asia/Shanghai"
```

注意：修改/etc/OpenStack-dashboard/local_settings.py 文件时强烈建议先搜索参数名，然后再来修改参数的值，比如在 VIM 中搜索参数名称 OPENSTACK_HOST，找到后把它的值改为“controller”即可。另外再检查

文件/etc/apache2/conf-available/OpenStack-dashboard.conf 有无如下一行，如果没有就添加：

```
WSGIApplicationGroup %{GLOBAL}
```

最后重新加载 Apache：

```
root@controller:~# systemctl reload apache2

# 或者直接重启 apache2
root@controller:~# systemctl restart apache2
```

浏览网址 http://controller/horizon 或者 http://10.7.70.45/horizon 即可访问 OpenStack 的 Web 管理页面，具体请参考 8.1.4 小节，这里不再赘述。

8.3 OpenStack 管理

我们已经采用 OpenStack 成功部署了一个三节点的私有 IaaS 云端，现在只有一个 Default 域，且 Horizon 默认不支持多域，一个域对应一家使用云服务的企业，而域中的一个项目（project）对应一个部门。域用来做名字隔离，项目用来做资源隔离。OpenStack 云端、域和项目的关系如图 8-16 所示。

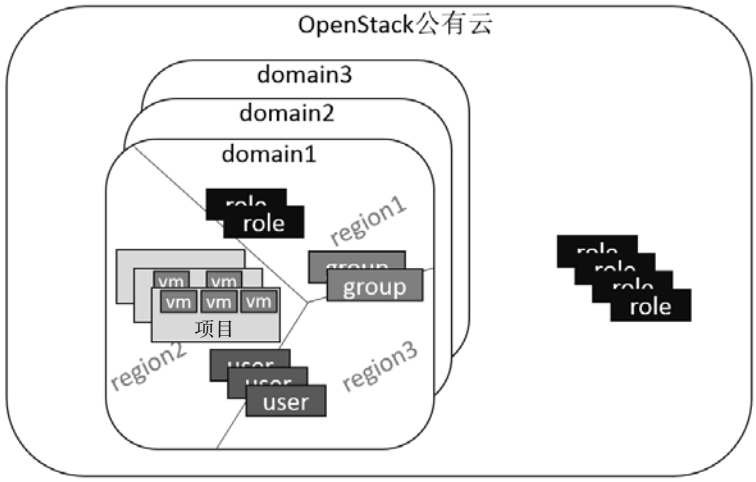


图 8-16　OpenStack 云端、域和项目的关系

私有云中存在域管理员和项目管理员，而公有云中还存在云管理员，云管理员一般由云提供商内部的人员担任，域管理员由租户（使用云服务的企业）内部人员担任，而项目管理员一般由租户内部的相应部门中的人员担任。例如深圳微算技术有限公司使用了重庆慧献云服务提供商的云服务，重庆慧献云管理员为深圳微算创建一个域 weisuan，并授权微算公司员工小王担任 weisuan 域管理员。然后小王在 weisuan 域中分别给开发部、市场部、生产部各创建一个项目，取名为 project_dev、project_sales、project_manu，并委托部门中的某个员工充当项目管理员。各个部门的项目管理员再给本部门的员工创建虚拟机，例如市场部的项目管理员小李给本部门的员工创建虚拟机。如果要采用 OpenStack 部署公有云，显然要开启多域支持。

OpenStack 管理涉及四类人员，云管理员、域管理员、项目管理员、普通用户，其分工如表 8-1 所示。

表 8-1　OpenStack 四类人员的工作分工

工种 / 人员	基础配置	日常工作	问题处理
云管理员	域级默认配置 物理设备管理 提供商网络配置 资源池化 镜像文件管理 制订规章制度	域管理 性能监控 合规性检查 域级资源配额 全局角色管理	全局级问题处理
域管理员	项目级默认配置 资源申购与释放 域内规章制订 与云管理员和项目管理接口	项目管理 用户管理 组管理 项目级资源配额 域内角色管理 费用检测与结算	域范围内的问题处理
项目管理员	配置资源模板 资源申购与释放	权限管理 虚拟机管理	项目范围内的问题处理

续表

工种 / 人员	基础配置	日常工作	问题处理
普通用户	偏好设置 申请虚拟机 申请虚拟网络	启/停虚拟机 使用虚拟机	虚拟机内部的问题处理 自助网络问题处理

在我们的实验环境中，Default 域管理员为 admin，密码是 ADMIN_PASS，打开网址 http://10.6.60.45/horizon，用 admin 登录之后的界面，如图 8-17 所示。

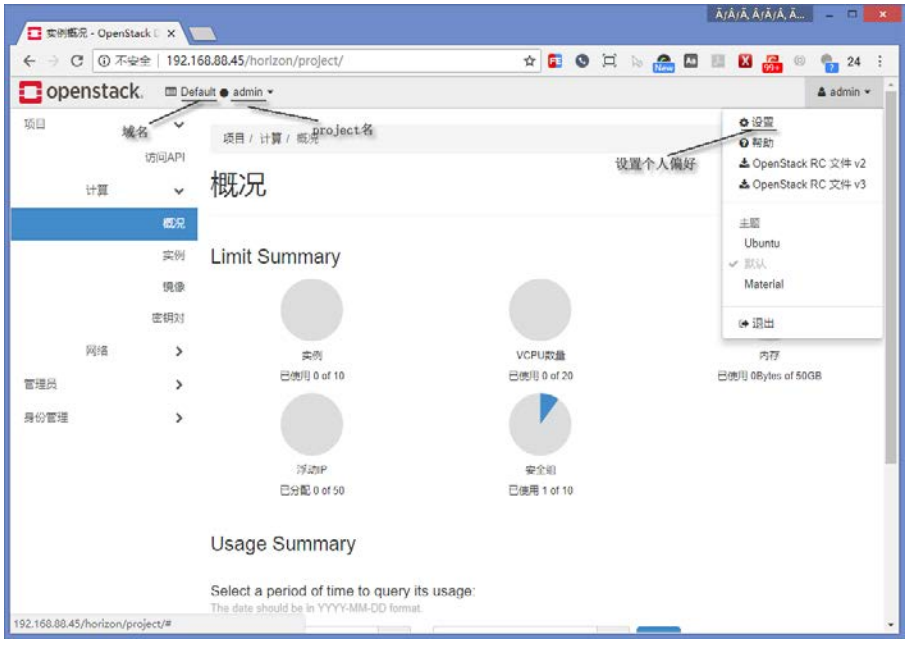


图 8-17　Horizon 的管理员界面

单击左侧的“身份管理”→“用户”，再单击“创建用户”按钮，给 myProject 项目创建一个管理员，如图 8-18 所示。

图 8-18　创建用户

然后退出 admin，再用 pro_admin 登录，因为 pro_admin 是项目管理员，所以无权做域级操作（比如创建、删除项目）。下面来创建一台 OpenStack 虚拟机，单击左侧菜单“项目→计算→实例”，然后再单击“创建实例”按钮：虚拟机取名 test_vm，源为 cirros，实例类型 m1.nano，选择两个网络 provider-net1 和 selfservice-net2，其他参数保留，最后单击“创建实例”按钮。如图 8-19 所示。

单击 test_vm，然后单击“控制台”即可通过 Web 界面登录虚拟机，如图 8-20 所示。

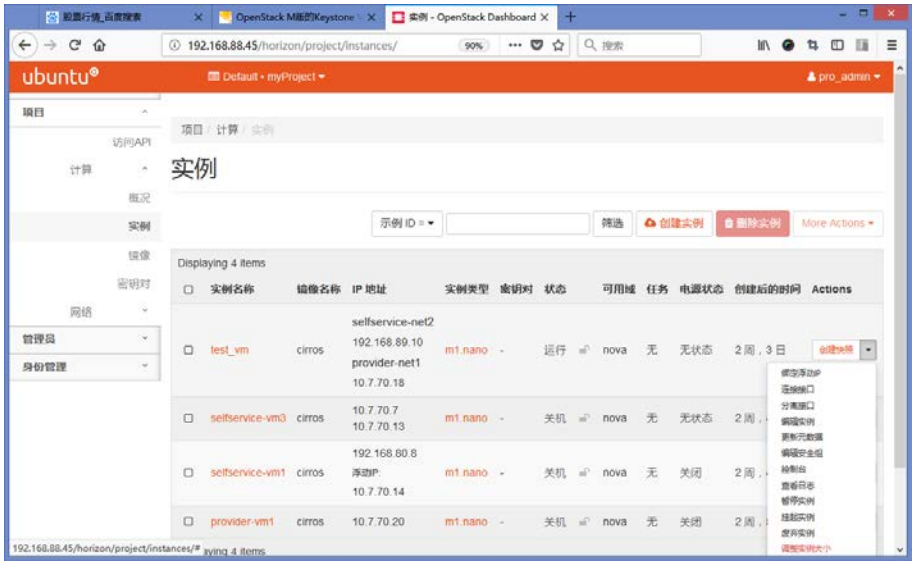


图 8-19　创建的虚拟机

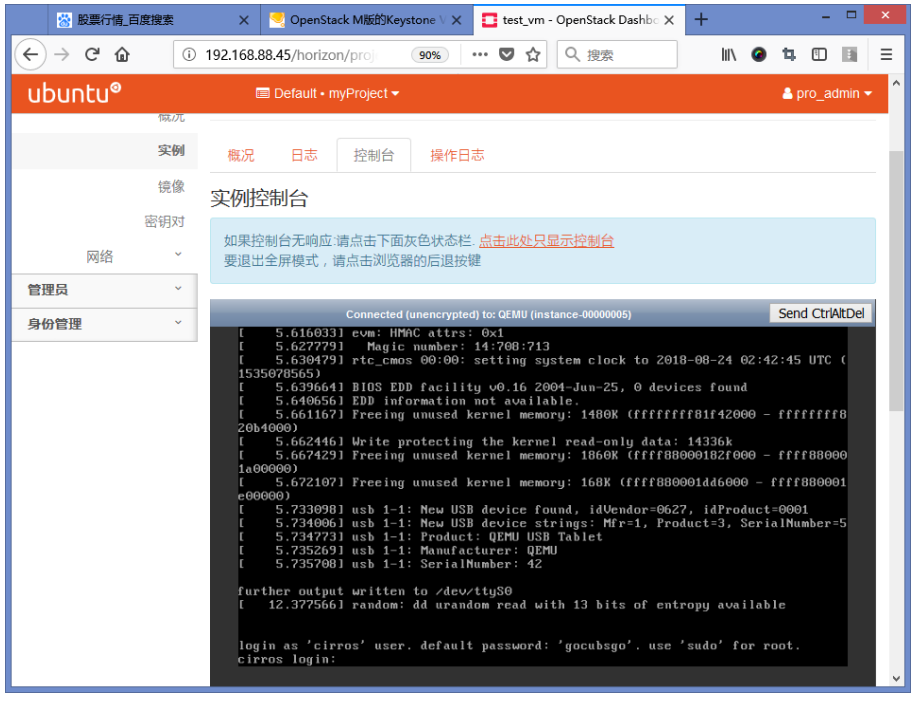


图 8-20　虚拟机的 Web 控制台

下面我们再来创建一台虚拟机并安装 Windows 7 操作系统。首先在本地计算机的硬盘上备好 Windows 7 的安装镜像文件（比如此处的 ISO 文件是 G:\isos\cn_windows_7_ultimate_with_sp1_x64_dvd_u_677408.iso），然后以 admin 登录 horizon 创建 Windows 7 安装镜像（管理员→计算→镜像），单击“创建镜像”，输入的参数如图 8-21 所示。

接下来创建虚拟机类型，单击菜单“管理员→计算→实例类型”，单击“创建实例类型”，如图 8-22 所示输入各个参数。

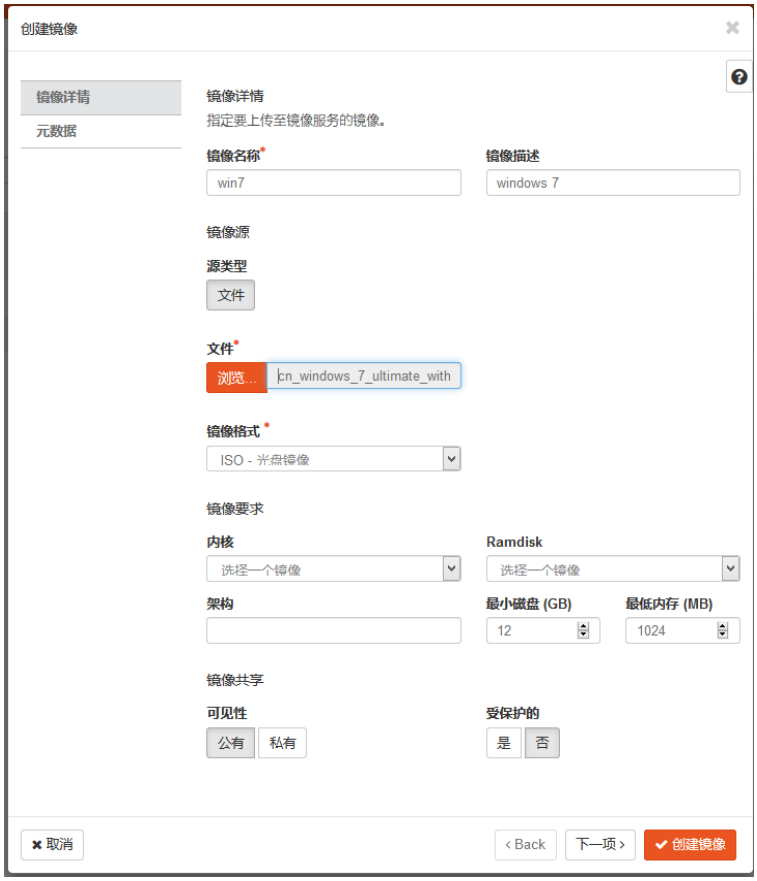


图 8-21　创建 Windows 7 安装镜像

图 8-22　创建 Windows 虚拟机类型

以 pro_admin 用户登录 horizon，查看当前空闲资源情况（菜单“管理员→虚拟机管理器→计算主机”），如图 8-23 所示。

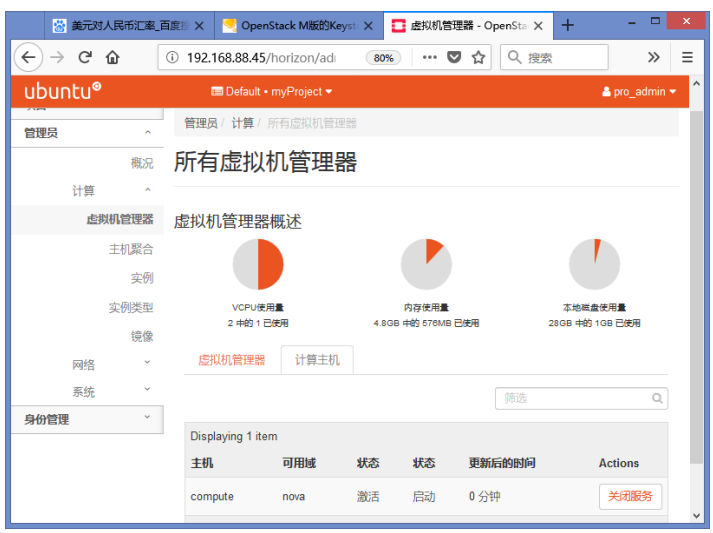


图 8-23　查看空闲资源

发现有一个空闲VCPU、4GB空闲内存、27GB空闲硬盘，足可以再创建一台 Windows 7 虚拟机了，因为在图 8-22 中创建的 Windows 虚拟机类型资源要求是 1VCPU、1.5GB 内存、16GB 硬盘。如果空闲资源不能满足即将创建的虚拟机的资源要求，就会报错“No valid host was found”，表示找不到满足条件的计算节点。

在计算节点上禁用 ubuntu 操作系统的 AppArmor 安全机制，这样就不会给 Windows 操作系统注入 AppArmor，因为 Windows 系统不支持它。

```
root@compute:~# apt-get purge apparmor apparmor-utils -y
root@compute:~# shutdown -r now
```

创建一台安装 Windows 7 的虚拟机。单击菜单“项目→计算→实例”，再单击“创建实例”，按照图 8-24 所示配置虚拟机参数。

图 8-24　配置虚拟机参数

单击刚刚创建的虚拟机，并打开控制台，在这里安装 Windows 7，如图 8-25 所示。

图 8-25　在虚拟机里安装 Windows 7

此后安装 Windows 7 就与平常一样操作，这里不再赘述了。安装完之后，可以打开远程桌面功能，这样就可以从其他计算机以远程桌面方式登录虚拟机了。